4

A RAND NOTE

Overview of System Software in the RAND Strategy Assessment System

Paul K. Davis, H. Edward Hall

December 1988

The research described in this report was sponsored by the Director of Net Assessment, Office of the Secretary of Defense (OSD), under RAND's National Defense Research Institute, an OSD-supported Federally Funded Research and Development Center, Contract No. MDA903-85-C-0030.

Published by The RAND Corporation
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138

A RAND NOTE

N-2755-NA

Overview of System Software in the RAND Strategy Assessment System

Paul K. Davis, H. Edward Hall

December 1988

Prepared for
The Director of Net Assessment,
Office of the Secretary of Defense

A research publication from
The RAND Strategy Assessment Center

40 Years 1948-1988

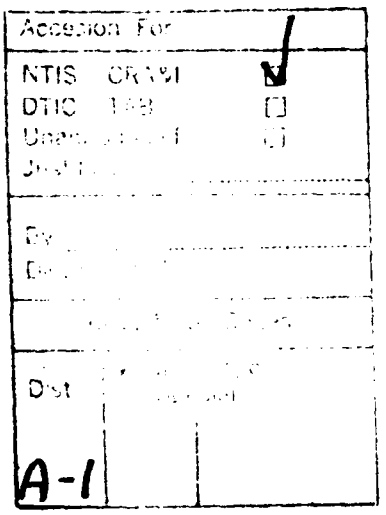
RAND

PREFACE

This Note documents and updates a briefing first given in January 1988 on the system-software aspects of the RAND Strategy Assessment System (RSAS). It reflects RSAS changes incorporated as of August 1988 for release in the RSAS 3.5. It assumes familiarity with the RSAS and was developed as only one part of a series of briefings and publications describing the RSAS and related analytic methodology. The bibliography suggests publications that should be read first. As is usual with briefing Notes, there is no attempt here to be comprehensive or detailed.

The work described here was conducted by the RAND Strategy Assessment Center, which is part of RAND's National Defense Research Institute, a Federally Funded Research and Development Center sponsored by the Office of the Secretary of Defense. The work has been part of a continuing effort to develop and improve the RSAS under a project sponsored by the Director of Net Assessment in the Office of the Secretary of Defense.

Dr. Davis is the Director of the RAND Strategy Assessment Center. Mr. Hall has been the project leader for RSAS system software development and can answer difficult technical questions that relate more to software than to RSAS modeling architecture, RSAS status, or applications. Both authors can be contacted at RAND in Santa Monica, California. Their electronic-mail addresses are: pdavis@rand.org and edhall@rand.org.



SUMMARY

The RAND Strategy Assessment System (RSAS) was designed for analytic war gaming. It is structured like a human military-political war game, but can be used in a number of different modes that include human play with all higher-level decisions introduced interactively, closed model operations in which the entire war game is played out as a simulation with no interventions, mostly closed operations in which an analyst or players interrupt the simulation to make selected decisions or corrections, and various hybrid modes such as that in which a human Blue team plays against an automated model representing Red (the "Red Agent").

This ~~briefing~~ Note describes RSAS system software. It is a reasonably rigorous description suitable for those concerned not only with the RSAS models but with their implementation as computer programs in a large and complex system. Users of the RSAS should also find it useful, because many of the complications encountered by those users can only be understood from a knowledge of software design.

REQUIREMENTS FOR RSAS SYSTEM SOFTWARE

Most of the requirements determining the system-software design were decided upon by RAND based on general guidance from the sponsor, extensive conceptual modeling, and certain visions of what technology should permit. This freedom to develop coherent requirements was a major factor in the project's success.

The *conceptual model* called for the RSAS to be structured as a military-political war game in which models called Red, Blue, and Green "Agents" represent the Soviet Union, the United States, and third countries, respectively, and in which a "Force Agent" (also called CAMPAIGN) simulates physical processes resulting from decisions by Red, Blue, and Green (e.g., processes such as the alerting, deploying, and combat operations of military forces). Red and Blue Agents are hierarchically structured to correspond approximately to real-world command-control structures with political-level authorities and a military hierarchy of commands translating objectives and strategy into operations plans. The political-level models have a strategic world view and can make sweeping decisions involving objectives, strategy, and escalation. The military-command-level models have more circumscribed behaviors; they attempt to follow "war plans"—adaptively, to be sure, but strongly constrained by guidance.

The conceptual model also called for a *globally integrated system* that would allow users to see interactions across various boundaries of theater, type of warfare, military service, and type of weapon—the very interactions that most models seek to approximate away in the search for simplicity and division of effort.

Some of the other important requirements decided upon early in RSAS development included an emphasis on *extreme flexibility* in many dimensions: the ability to substitute humans and agents for one another, allowing for both gaming and analytic modes; the ability to change a vast array of assumptions readily by varying not only parameters but the underlying rules and algorithms as well; and providing alternative perspectives and to some degree alternative levels of resolution. The system also had to be functionally modular, since different users would wish to focus on different nations, theaters, and types of conflict.

Another fundamental requirement was that of achieving as much *transparency* as possible within the state of the art. For example, it should be possible for nonprogrammer analysts (and to some extent senior officials) to review, understand, and change key assumptions. Toward this end there was a conscious effort to exploit newly emerging computer technology, including workstation technology and various concepts about high-level programming languages understandable to nonprogrammers.

There was also a requirement for a high level of system *performance*, because a major objective was to permit extensive sensitivity analysis. We sought (and achieved) run times of a few hours for multitheater wars with operation of both decision and simulation models, and a few minutes for simulation of combat in a single theater over a period of perhaps 30 days. Finally, we required *portability*, because it was correctly anticipated that the RSAS would be used in a variety of government agencies as well as RAND. Although the models and most system software are indeed portable, the graphics software depends currently on special features of Sun™ workstations, which are now rather inexpensive.

MAJOR FEATURES OF THE SOFTWARE APPROACH

Some of the most important features of the software design included: (1) using hierarchical coprocesses to represent the natural hierarchical objects of the conceptual model; (2) using a centralized data base for agent-to-agent communication (in preference, for example, to message-passing in the sense of object-oriented programming languages) ; (3) building a new interpretive language for knowledge-based modeling (RAND-ABEL®), (4) using the C language for the numerically intensive and bookkeeping- and process-dominated Force-Agent simulation; (5) special capabilities for saving and branching, and for conducting

"lookaheads" within the course of a game run; (6) various special mechanisms for control flow; and (7) exploitation of workstation technology.

To exploit workstation technology, we have used not only the multiwindow features and mouse techniques of the Sun workstation, but have also developed a number of useful general-purpose software tools for graphics, cross-referencing of variables, logging explanations, examining and changing the source code, and allowing the analyst to examine and change the values of parameters interactively. Thus, we have developed an entire *environment* for knowledge-based modeling and simulation on workstations.

FUNCTIONAL DESCRIPTION

- The RSAS can be operated in a variety of modes: (1) supporting a human war game in which players make all key decisions interactively and combat models are then used to assess the results of those decisions; (2) as a closed or systemic model in which decision models called agents substitute for human players; or (3) as a hybrid in which, for example, a Blue human team plays against an automated Red Agent.
- The knowledge-based decision models or human players issue orders to military forces worldwide; a simulation model called Force Agent (or CAMPAIGN) then acts upon these decisions; it is itself a large, complex, and integrated software system consisting of models for many different types of warfare and geographical theaters representing processes such as alerting, deployment, maneuver, and combat. Force Agent is structured largely as a two-player game, so the force orders of nonsuperpowers are handled through Red and Blue.
- The Force Agent is a time-step simulation, with the time-step duration varying depending on conditions in the simulation.
- In its automated mode the "game" has as its decisionmaking entities Red, Blue, and Green Agents, representing, respectively, the Soviet Union, the United States, and nonsuperpower countries. The RSAS is an n-player game at the political level.
- Red and Blue are composite models consisting of hierarchies of agents corresponding approximately to the National Command Authority and several layers of military commands. Green is a one-level composite model with submodels for each of the nonsuperpower nations being treated.
- Each of the decisionmaking entities or agents makes its decisions independently, although constrained by guidance from above in the instance of subordinate military commanders. Each agent has the opportunity to act whenever its "wakeup rules" fire—that is, when conditions specified by wakeup rules and tested by the simulation model are met. The conditions may be specified in terms of time, events, or other measures of the world state.

- Agents at different levels of the Red or Blue hierarchy can have quite varied characters. At the top level, decisionmaking reflects a strategic world view and can involve fundamental changes in objectives and strategy; at lower levels, decisionmaking corresponds to military commanders attempting to carry out highly procedural plans (albeit branched and otherwise adaptive plans).
- The agents can conduct lookaheads to better assess alternative courses of action. A lookahead consists of a game within a game in which, for example, the Red Agent uses the RSAS to predict what may happen under a postulated strategy. The lookahead depends on Red's assumptions about the behavior of Blue, nonsuperpower countries, and various parameters of the simulation. That is, the RSAS includes not only Red, Blue, and Green but also alternative versions of these agents representing superpower assumptions—for example, Red's Blue.
- The RSAS is extremely flexible in terms of the parameterization of models and the ability of users to change the basic decision rules interpretively. Moreover, a special agent acting on behalf of the analyst (Control Agent) can be used to change parameters in the course of a game if and when conditions specified in rules arise—just as the analyst might otherwise follow the game step by step and interrupt it to make changes when he saw the conditions arising.

SOFTWARE CONSIDERATIONS

- The decision-modeling portion of the RSAS consists of *quasi*-independent agents, with Red and Blue having several hierarchical levels within their agents, and Green having a single level. Each level of an agent is implemented by one or more *coprocesses*—a sort of program within a program. Although coprocesses execute independently, they can be created or removed by higher levels of the hierarchy, and communicate with each other via a shared data base. Thus the RSAS is a variable-structure simulation: e.g., parts of the Red or Blue Agent hierarchy are replaced if the situation warrants it, such as an escalation from conventional to tactical nuclear warfare.
- This process implementation provides for modularity, "script-like behavior" by some of the military-level models in accord with the plan-following style of the conceptual models, and the flexibility to easily change command structures or, within a given run, to change the entire block of rules used to represent a given agent.
- The decision models all draw on and contribute to a centralized data base. Modularity is accomplished by access restrictions incorporated in the language and a Data Dictionary. Thus, the model for a particular Red command cannot use the internal variables of other Red commands or, much less, Blue commands, except by using a special syntax that highlights the violation of usual modularity. Message-passing is accomplished by one agent writing to the data base and the appropriate recipient reading that message, again with access restrictions. This has both advantages and disadvantages over the message-passing facilities provided in object-oriented programming languages.

- The rules constituting the decision models are written in a new readable procedural language for knowledge-based models, RAND-ABEL. By far the majority of the rules are expressed in RAND-ABEL code as decision tables or tables of orders. Because RAND-ABEL is understandable and interpretive (the latter facilitating a gradual learning process), analysts who are not good programmers can review and change the decision rules quickly during or between runs, without recompiling. The RSAS simulation model is written largely in C, and partly in RAND-ABEL, which translates into C.
- The RSAS is substantially modular in allowing users to focus on particular problems of interest to them without necessarily operating the full global system. For example, users may turn the decision models off and substitute simple sets of hard-wired instructions as data prepared by filling out spreadsheets. They may then focus on the combat simulation for one or more theaters. Alternatively, they may operate the political models (national-command-level models) by themselves, specifying inputs interactively that otherwise would come from the simulation model.
- RSAS system software includes an integrated environment for both developing decision models and conducting analytic war games and more conventional simulations. This environment includes C and RAND-ABEL working under the UNIX™ operating system, and provides a mouse-and-menu-driven data editor, nested menus for accessing the source code and data bases, an interpreter for RAND-ABEL, cross-indexed source code, variable-resolution logs, graphics, and other tools. Some of the graphics tools have been specialized to Sun workstations.

STATUS

The RSAS is now operational and is being used in a number of different applications ranging from training-related and concept-generating gaming to analysis of alternative military strategies and arms-control concepts. The RAND-ABEL language has proven highly successful, and is used routinely by analysts who would find it counterproductive to work at the level of computer code in more conventional languages. The larger environment that includes not only RAND-ABEL but also the various software tools mentioned above is now being used in other application domains and is being prepared (under the name RAND-ABEL Modeling Platform, or RAMP) for general release in the public domain. RAND-ABEL is also being extended to include a variety of features such as sets, lists. and structures. Although the current RSAS has many imperfections and there is a long list of additional or different features we might like, we believe nonetheless that the RSAS will be viewed from a historical perspective as having set new standards and introduced a variety of new ideas for variable-structure knowledge-based simulation in complex problem domains. We

expect both the RSAS and RAMP (including RAND-ABEL) to evolve gracefully and significantly in the years ahead. Some of the more important software improvements will include better integrating the RAND-ABEL and C language portions of the system, adding tools for software engineering (e.g., tools to generate lists of module inputs and outputs), extending the RAND-ABEL language to include features useful in implementing object-oriented models, and adding tools for pre- and post-processing.

Contents

OVERVIEW OF RSAS SYSTEM SOFTWARE

Paul K. Davis and H. Edward Hall

December 1988

A Briefing Note from the RAND Strategy Assessment Center

KEY FIGURES IN SYSTEM SOFTWARE

Project Leader: Edward Hall

System Software Design: Edward Hall, Norman Shapiro, Arthur Bullock

RAND-ABEL language: Norman Shapiro, Edward Hall, Mark LaCasse, Robert Weissler

Systems Programming: Edward Hall, Arthur Bullock

Interfaces: Jean LaCasse, Robert Weissler, Barry Wilson, Mark LaCasse, Mark Hoyer, Sharyne Blixt, David Frelinger, Stuart Glennan

Graphics: Larry Painter, Mark LaCasse, Robert Worley

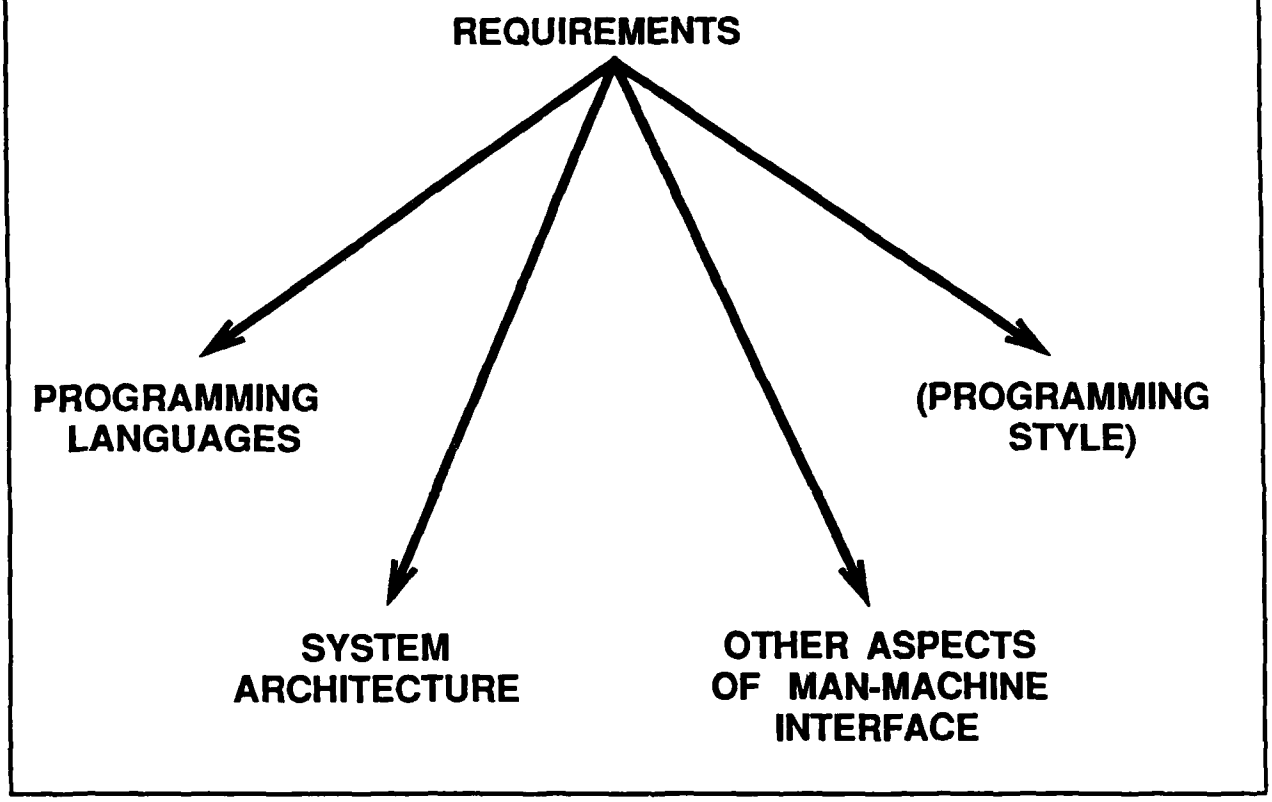
Control Programs: Christe McMenomy

Management: Paul Davis, Herbert Shukiar, Bruce Bennett, William Schwabe

KEY FIGURES IN SYSTEM SOFTWARE

The development of RSAS system software has been a multiyear effort and will continue for some time into the future because of the many opportunities to greatly improve man-machine interfaces. One of the authors (Edward Hall) has led the system-software development effort. In earlier years, Norman Shapiro played a seminal role in initial design and was the principal designer of the RAND-ABEL language. Arthur Bullock developed the initial structures used in the simulation-model part of the RSAS and has been a major contributor to overall system-software development since then. As the chart indicates, there have been many other contributors to this multipartite effort. Herbert Shukiar, Bruce Bennett, and William Schwabe should also be mentioned for their managerial influence on many aspects of design and development. Herbert Shukiar was also kind enough to review the draft of this briefing Note.

TRANSLATING REQUIREMENTS INTO DESIGN



TRANSLATING REQUIREMENTS INTO DESIGN

The briefing proceeds from "requirements" (mostly self-imposed) into design considerations, choice of approach, and finally design details. For expositional reasons such as the need to introduce terminology, we discuss language issues first. We then go into architecture more generally and into the man-machine interface. Although we don't discuss it here, dictates of the project had important effects on programming style. In particular, we adopted structured programming and decided to emphasize on-line documentation and intuitive interfaces wherever possible, in preference to producing masses of traditional hard-copy documentation that would probably seldom be read.

CONTENTS

CONTENTS

Let us begin, then, with a discussion of "requirements" for the design of system software.

CONSIDERATIONS IN SYSTEM SOFTWARE DESIGN

REQUIREMENTS FROM MODEL ARCHITECTURE

- Structure of a military and political war game
- Hierarchical agents/players as natural objects of game
- Two distinct types of decision modeling:
  - National command models with strategic world view
  - Analytic war plans as adaptive building-block scripts
- Scheduled and unscheduled action triggers
- Lookaheads with imperfect information (e.g., Red's Blue)
- Variable-resolution time steps
- Extreme flexibility: human/agent substitution, closed and interactive operations, analytic and gaming modes, changing of parameters, rules, and algorithms; alternative levels of play affecting perspective and resolution
- Reproducible and understandable results

CONSIDERATIONS IN SYSTEM SOFTWARE DESIGN

Our system software requirements were largely established by early 1983 (Davis and Winnefeld, 1983), even though many important modeling ideas emerged later (e.g., see Davis, 1985). The aspects of modeling approach that were especially significant in determining software requirements were: (1) the very concept of the automated war game (due primarily to Carl Builder and William Jones; see Graubard and Builder, 1980); (2) the need for hierarchies of agents; and (3) the need to accommodate two different styles of agent decisionmaking: a strategic style of decisionmaking in which the top-level (political) model should to some extent look at all variables and be able to make radical changes of strategy if necessary, and a more cybernetic style by which the subordinate (military-level) models would be adapting on the margin within the confines of a plan established at a higher level.

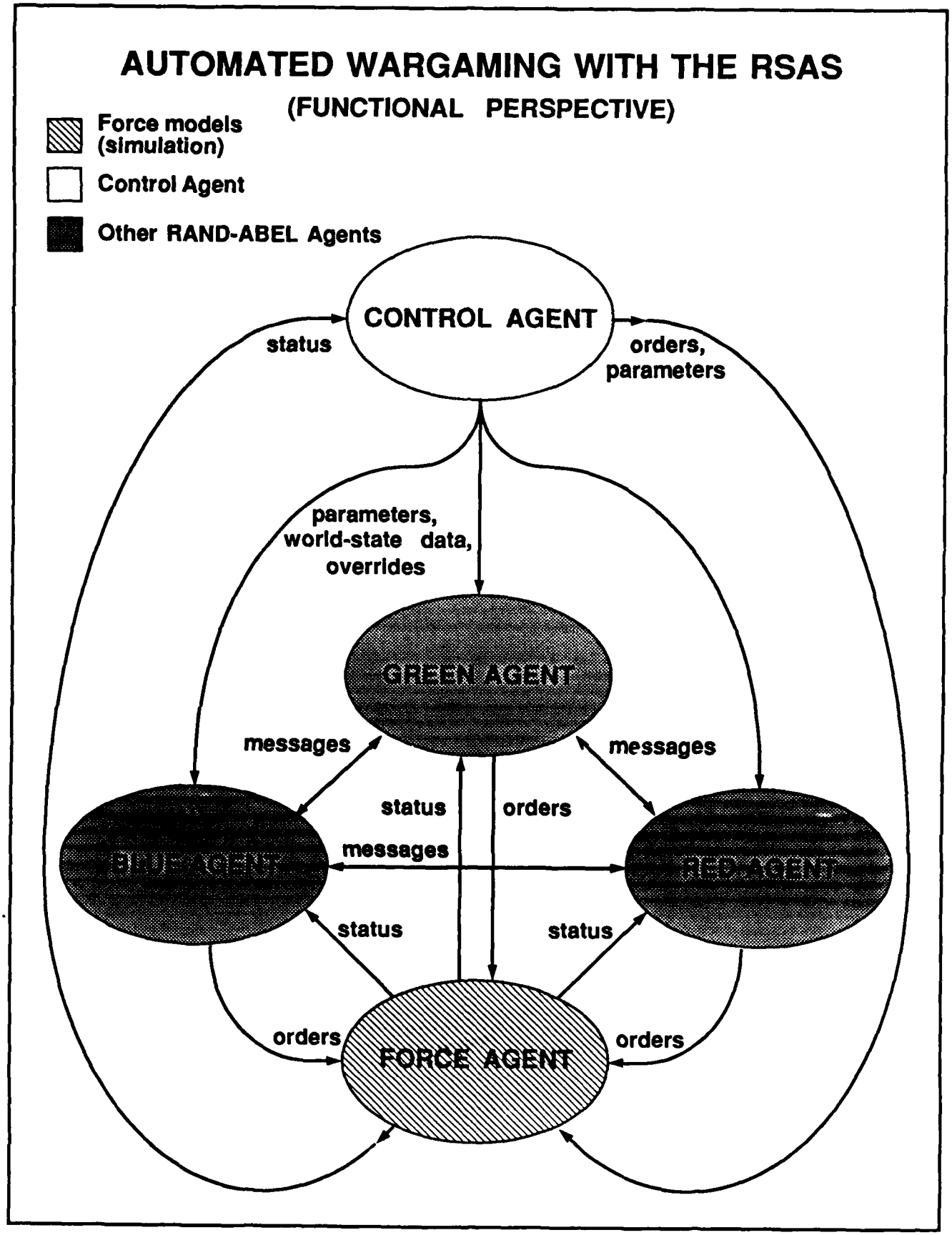
The notion of conceiving of military-level decisions as being adaptive only within the confines of an analytic war plan akin to the scripts of artificial intelligence was especially important—in part because of the conceptual paradigm involved, but also because it demanded a radically different approach to the programming itself, as we shall discuss shortly.

Other key requirements included: the need for agents' actions to occur at scheduled times or when certain conditions arose, the use of lookaheads (a game within a game, conducted to help an agent make a decision), and variable-resolution time steps.

An important aspect of the requirements was the emphasis on transparency and ease of use. It is worth mentioning that many observers over the years have seen relatively little value in some of this, because they seek a black-box model amounting to an "answer machine." However, we took the viewpoint that assumptions and their interrelationships drive conclusions, and it is therefore desirable that analysts and senior officials be able to get definitive explanations and have the opportunity to change such assumptions readily. Moreover, from early on we envisioned a system that would be used by the analyst himself in the course of his exploration and creative thinking, and not merely as a production device. Our image, then, emphasized interactiveness, flexibility, and as much user-friendliness as we could manage.

AUTOMATED WARGAMING WITH THE RSAS

(FUNCTIONAL PERSPECTIVE)



AUTOMATED WARGAMING WITH THE RSAS

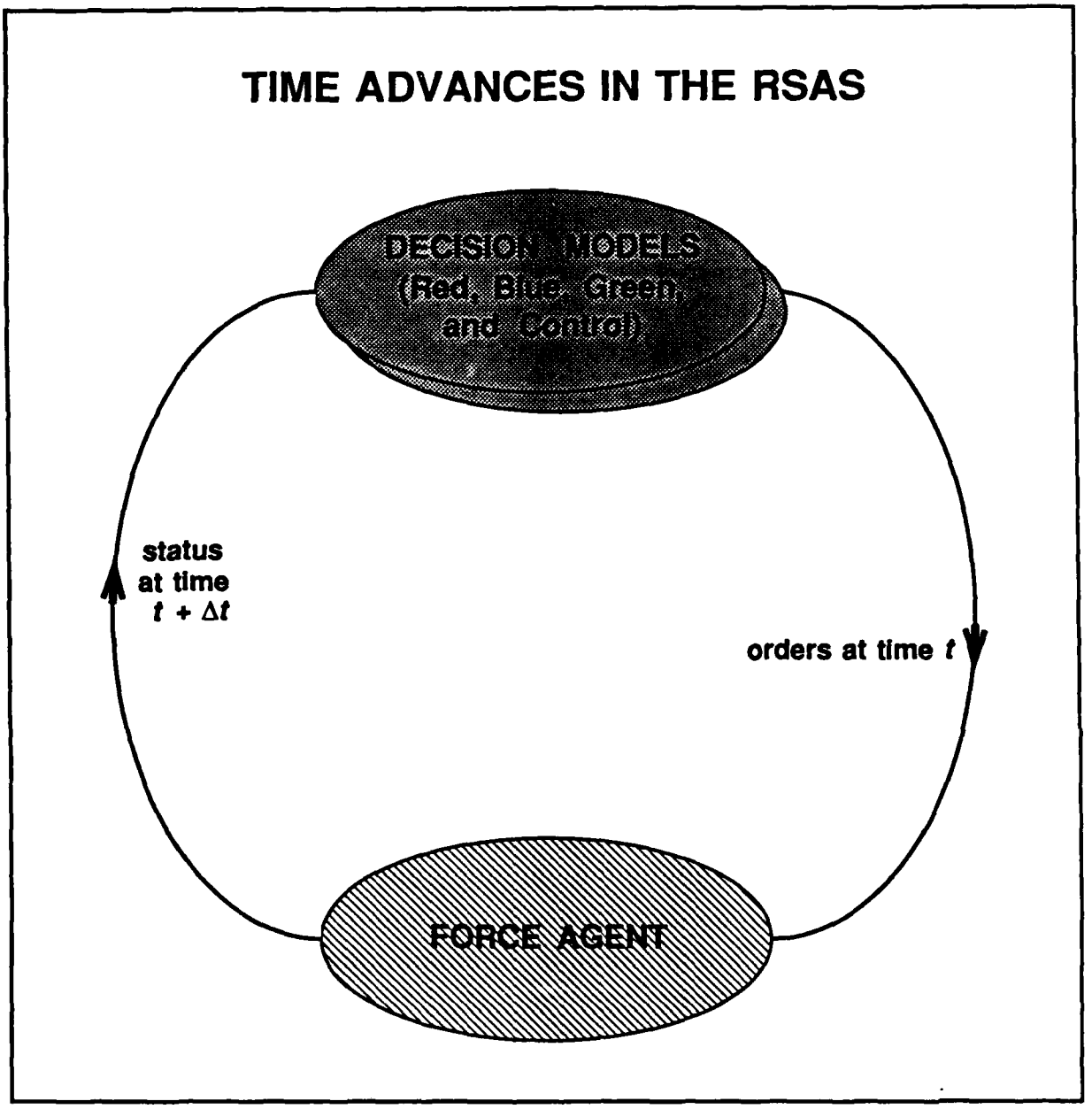
Elaborating on the concept of automated wargaming, this chart and the next two illustrate certain key features of model architecture as described elsewhere (e.g., Davis, 1985). It is important to distinguish between the conceptual *model* and the implementing computer *programs*. In principle it is possible to implement a given model in many different ways. Which way one adopts affects performance, transparency, maintainability, flexibility, and a multitude of other factors. Here we see the top-level view of the RSAS with its five basic agents, of which Force Agent (CAMPAIGN) is special by virtue of being responsible for *simulating* the physical world—i.e., for modeling how the physical world changes over time. These changes are in response to decisions and directives issued by the other agents, which are *decision models* rather than *simulations* (although in some fields of study "simulation" has a more general meaning that may include nontime-dependent modeling, or even the acting out by humans of possible discussions, arguments, and decision processes).

The Force Agent receives orders and produces information on the world state. It is in itself a complex and sophisticated complex of models (see Bennett, Jones, Bullock, and Davis, 1988), but will be treated in this Note as a black box. That is, we are mostly concerned here only with how Force Agent fits into the overall system—although we make some observations from time to time about programming methods used for the Force Agent.

Note that the political models must communicate (i.e., nations send messages to each other during crisis and war—asking for cooperation, offering termination conditions, and so on). Moreover, as shown on the chart GENERIC STRUCTURE OF RED AND BLUE AGENTS (which follows the next chart), the conceptual model requires, consistent with the real world, that there be a hierarchical command-control structure within the Red and Blue Agents, and that there be messages up and down the command line. These are distinct from the "orders" that the various models representing political- or military-level commands would issue to the military forces, which would result in physical activities (e.g. alerting, deploying, and dispersing armies or air forces). The simulation model (Force Agent) cares only about the orders, except that the simulation is parameterized in many dimensions and the Control Agent (defined as the automated agent serving for the analyst) may change those parameters from time to time.[1]

---

[1]As a side note, an agent conducting a lookahead may also change those parameters, and parameters affecting other agents, so that the lookahead is based on the agent's *perceptions* of the world rather than reality.


TIME ADVANCES IN THE RSAS
DECISION MODELS (Red, Blue, Green, and Control)
status at time $t + \Delta t$
orders at time t
FORCE AGENT

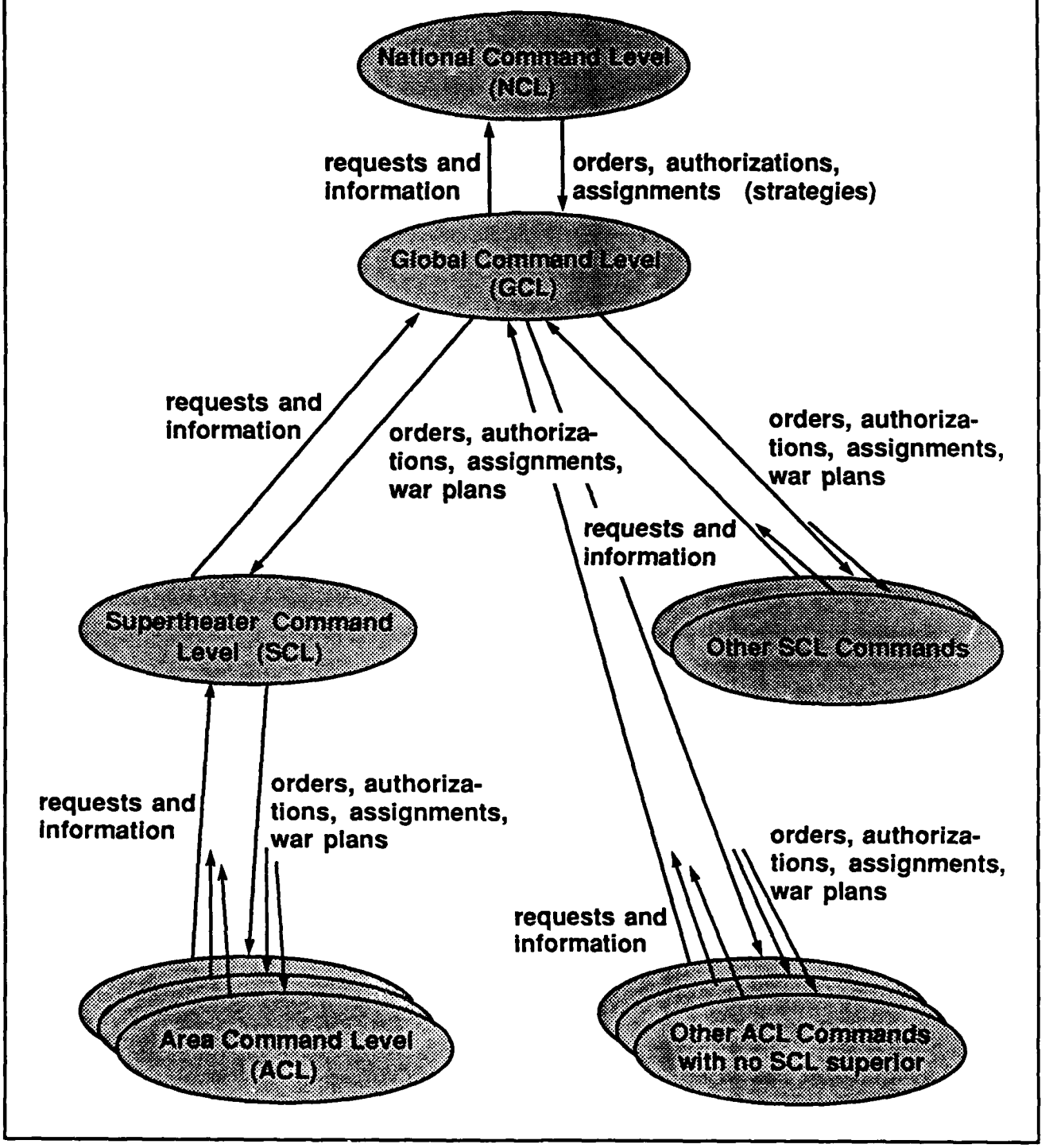
TIME ADVANCES IN THE RSAS

This chart illustrates how time is handled within the RSAS. Since the Force Agent is responsible for simulating the development of the world state over time, it is natural to make it responsible as well for controlling the simulation clock. Thus, simulated time advances only when Force Agent dictates such advances. By contrast, while the Red, Blue, Green, and Control Agents (or human players) are deliberating, simulated time is frozen. The decision models are responsible for specifying not only the decisions, but also any delay times that might affect those decisions. In some cases a decision model "holds its decision" for a while; in other cases ("flag model" cases) it sends the order at once but specifies when the action is to occur. From the viewpoint of software the decision models are entirely event-driven, whereas Force Agent is concerned with both events and continuous processes. Time advances are variable depending upon the situation (e.g., typically four hours for conventional combat but only minutes during a nuclear attack).

If the decision models are to be event-driven, they must specify the conditions under which they will act. That is, they must specify what we call *wakeup rules*, since each agent can be imagined to be *sleeping* until an event for which it wishes to take action occurs. Wake-up rules can specify conditions based on time as well as on changes in the world state; for example, when the Red Agent is fighting a war it may specify a deadline at which time it will re-evaluate its strategy if it has not made a sufficient degree of progress. Many delays are also implemented by the decision models having wakeup rules triggered at the time orders are to take effect. Agents are able to specify multiple wakeup rules; in the preceding example, Red also may specify a wakeup rule triggered by Blue nuclear use.

It should now be clear why this briefing does not emphasize control-flow within the simulation—it is highly dependent upon the progress of the simulation itself. However, we will return to the matter of wakeup rules and control flow later in the briefing.

GENERIC STRUCTURE OF RED AND BLUE AGENTS



GENERIC STRUCTURE OF RED AND BLUE AGENTS

This chart elaborates on the hierarchical concept. Both Red and Blue have a national-command-level (NCL) model that decides objectives and strategy (Davis, Bankes, and Kahan, 1986), and several levels of military-command models, developed principally by colleagues William Schwabe and Barry Wilson. There may be no or many supertheater commands, and each supertheater command may oversee many area commands. As an example, Blue's Global Command Level consists of the "JCS Model", which represents the role of the Joint Chiefs of Staff and certain functions of State and the National Security Council. Blue also has a supertheater command called EUR, which corresponds approximately to the real-world Supreme Allied Commander, Europe (SACEUR). Red has no supertheater command level currently, because the real-world Red apparently has none (i.e., theater commanders report directly to the national-level military command). There are subordinate Area Command Level (ACL) commands for Northern, Central, and Southern Europe. The military-level models have the character of adaptive plans, which we call *analytic war plans* to distinguish them from real-world operations plans, which seldom specify how to adapt to more than a very few contingencies.

Commands can communicate both up and down the line of authority. For example, an area command called AFCENT can send information to and make requests of the supertheater command EUR, which in turn looks upward to JCS, and so on. Orders, by contrast, go *down* the line. The *de facto* skipping of echelons can be represented by having intermediate levels pass messages along with no change or delay.

CONSIDERATIONS IN SYSTEM SOFTWARE DESIGN (CONT'D)

OTHER NEEDS AND DESIRES

- Globally integrated Force data base
- Fast run time (half day for full 30-day global simulation)
- Ease of use
- Eventual distributed operations
- Portability
- Reliability
- Maintainability

SYSTEM SOFTWARE DESIGN CONSIDERATIONS (CONT'D)

Here we complete the summary of the major considerations that served as generic requirements. Touching upon the items briefly:

- The RSAS was to be an *integrated global* model of crisis and conflict. It should be straightforward, for example, to have "strategic nuclear assets" such as KC-135 tankers used in conventional conflicts and then pulled back for their primary strategic nuclear missions, but users should not be able to double count them. Similarly, users should have to make choices about where and when to use the 82nd Airborne or strategic airlift. Also, if one set of actions implies bombing of certain installations, then the damage of that bombing should be accounted for when comparing other actions later on—even if the actions are in different compartments (e.g., conventional vs. tactical nuclear strategic nuclear targeting, or the targeting of military vs. C^3I installations).

  We elaborate on this because it had a profound effect on design. Ordinarily, one tries to modularize problems so that different workers can proceed without stepping on each other's toes or worrying about the details of other workers' tasks. For the RSAS, however, it was essential that the forces be realistically intertwined, which required a large and complex centralized data base, designed by colleague Arthur Bullock. We expected that managing such complexity would be difficult, and it has been. This is a major reason for our decision not to distribute source code for the simulation part of the RSAS, although users are welcome to review it at RAND and suggest changes that would be helpful to them.

- We sought fast run times, but were not able to estimate well what would be possible. On the one hand, we wanted narrow calculations and decision modeling to be very fast—within a user's cognitive cycle measured in seconds or tens of seconds. On the other hand, for a full-up global simulation we thought anything up to about half a day in duration would be both tolerable and a major achievement. Even though our numbers were soft, they implied the requirement for a new language that would be used for the decision models.

- It seemed that the nature of man-machine interfaces was changing, and that the RSAS should be designed so that analysts and wargamers could use the RSAS themselves with only moderate assistance from professional programmers. Simplifying the use of the RSAS has been exceptionally challenging because of the inherent size and complexity of the problems being treated.

- Although it seemed visionary in the early 1980s, it was clear to some of the system architects that the time would come when people would want to have the different agents or teams operating on different computers. This might arise for functional reasons or because of the desire to exploit the emerging technologies of local-area networks and small, inexpensive but powerful computers.

- Portability was considered essential because the RSAS was to be transferred to government agencies with varying hardware and software configurations. It was recognized, however, that "portability" should be defined in functional terms: the man-hours and capital expense to transfer the system from one installation to another.
- Reliability and maintainability, which often get short shrift, were fundamental issues because of the probable complexity of the RSAS. Concern for these matters dictated adherence (albeit sometimes imperfect) to modern computer science techniques often considered a bother by application modelers. The basic issue here is that difficulties increase geometrically with complexity and control techniques become critical, not optional, for large systems such as the RSAS.

These, then, represented the requirements around which the RSAS was to be designed and built. They were mostly self-generated and informal, but were for the most part reviewed and approved by a DoD Steering Group.

Probably the most important single difference between the RSAS and the dimly envisioned RSAS of the requirements period in the early 1980s has to do with the microcomputer workstations. Rather far along in the development process (1985), RAND decided to begin making the move to Sun workstations and away from systems such as a VAX™-11/780 minicomputer coupled with a text monitor and a separate graphics monitor. For some time RAND maintained the option for operations on such systems, and for some time there were severe criticisms of our move toward workstations, largely because many prospective RSAS users were just beginning to receive long-awaited and expensive VAX minicomputers, which they now hoped to use for the RSAS. By now, however, the jury is in: users love the workstation environment and their managers have finally recognized that the cost of workstations (and hardware more generally) is now low enough so that it is wise to pay the bills and get the hardware that makes efficient man-machine operations possible. The result has been that the first operational version of the RSAS exploited a technology that was still very new. We believe that delaying the transition to that technology would have ultimately cost the government a great deal of money and dramatically reduced the acceptance of the RSAS, despite the virtue of RSAS models.

CONTENTS

CONTENTS

Let us now turn to the issue of programming languages, because understanding this aspect of the RSAS is crucial to all that follows.

APPROACH TO PROGRAMMING LANGUAGES

- Combination: C and new RAND-ABEL translating into C
  - — C for computationally intensive, globally integrated force processes, exploiting pointers and structures
  - — RAND-ABEL for clarity, flexibility, rapid prototyping, and analyst accessibility
  - — RAND-ABEL for analytical knowledge-based modeling in complex systems, including many algorithmic modules

IN C LANGUAGE

Force-C Models (CAMPER)
- Global bookkeeping
- Strategic mobility
- "Main" theaters (Center Region, Korea)
- Most naval operations
- Most strategic nuclear
- Most space

IN RAND-ABEL LANGUAGE

Higher-level decision models
- Superpowers (Red, Blue)
- Third countries (Green)

Control Agent

Force-A (e.g., Referee)
- "Secondary" theaters (e.g., Iran, Norway)

Some other force models

APPROACH TO PROGRAMMING LANGUAGES

It soon became evident that we would need two different programming languages. To achieve a fast global simulation model of conflict it would be necessary to use a highly efficient general-purpose programming language such as C, but if the knowledge-based models were to be flexible and accessible to analysts something special would have to be developed for them. That something has turned out to be RAND-ABEL (sometimes loosely referred to as "Abel"), a versatile language described elsewhere (Shapiro et al., 1985, and Shapiro et al., 1988).

The decision to use the C language and UNIX environment was based upon several considerations (see also Shapiro et al., 1985). In particular, C/UNIX assured portability, a good development environment in terms of power and flexibility, and specialized tools such as YACC and LEX for building a high-level language for the decision models. Another important factor was that C/UNIX was already familiar to a number of RAND's computer scientists. ADA™ was briefly considered but rejected on various grounds.[1]

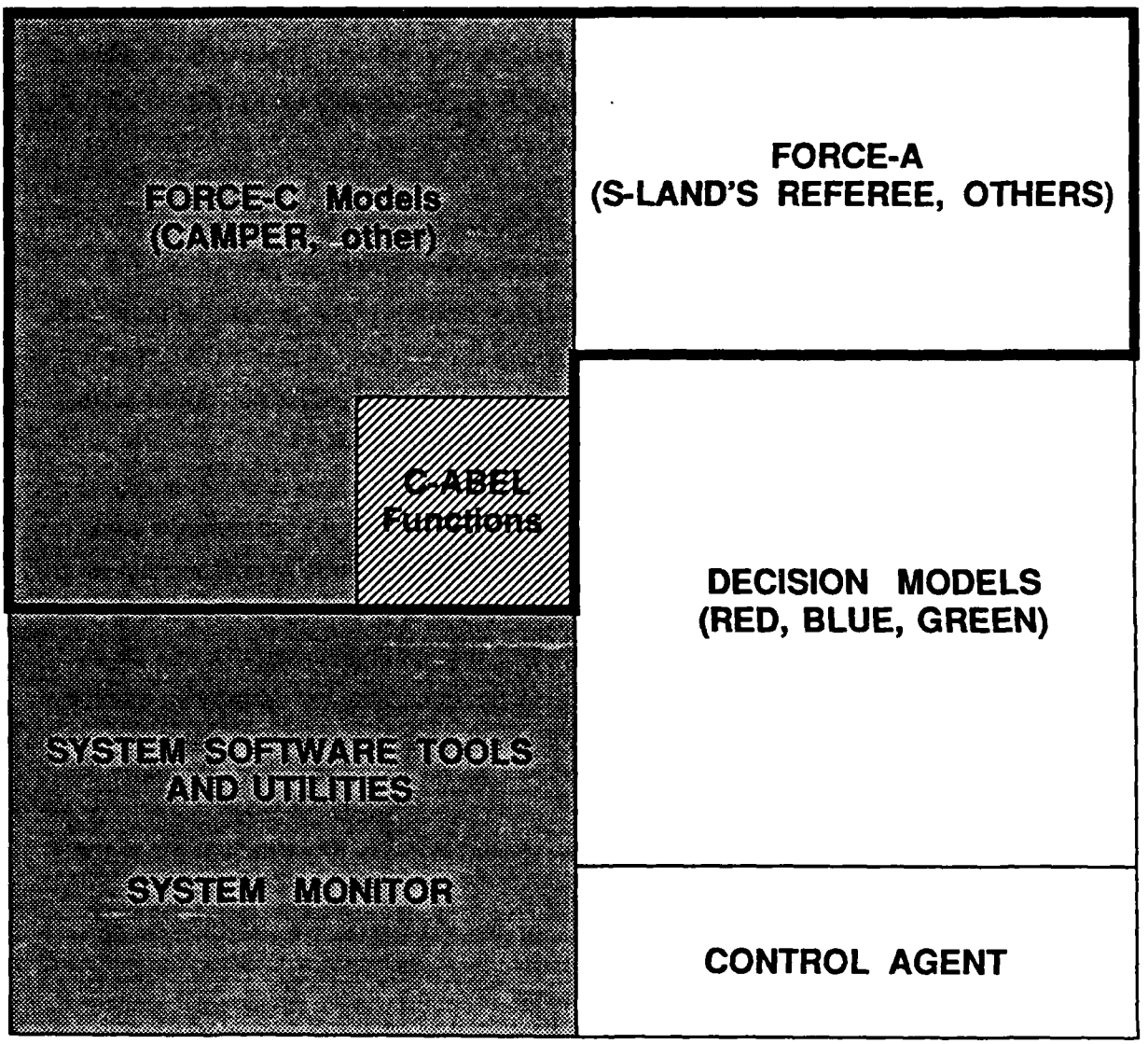
At the anecdotal level, it may be worth noting that some of our application programmers who previously worked in FORTRAN are of the opinion that the Force model could not have been built without something as powerful as the C language with its pointers and structures, and the various UNIX programming utilities. In any case, RAND's experience with the C/UNIX environment has been generally favorable. Long-term maintainability and transparency to new programmers are yet to be assessed, however. Better programming environments are assuredly needed to cope with such matters in complex systems.

---

[1]ADA was just becoming available at this time, 1983, and not on a hardware/software platform that RAND was likely to support or that the project could have afforded on its own. Two other factors entered as well: lack of in-house experience with ADA would have meant an extended delay in development, and ADA's capability for man-machine interface appeared weak to us. Thus it was considered a bad choice. Today we would consider ADA a reasonable choice for the Force simulation and a viable (but not as facile) alternative for the decision models, but since current ADA environments don't yet interface well to windowing systems there would still be problems with the man-machine interface.

RSAS COMPOSITION BY PROGRAMMING LANGUAGE

(NOT SCALED BY SIZE OF PROGRAMS)



C

RAND-ABEL

RAND-ABEL within
C program (C-ABEL)

Force Agent
(CAMPAIGN)

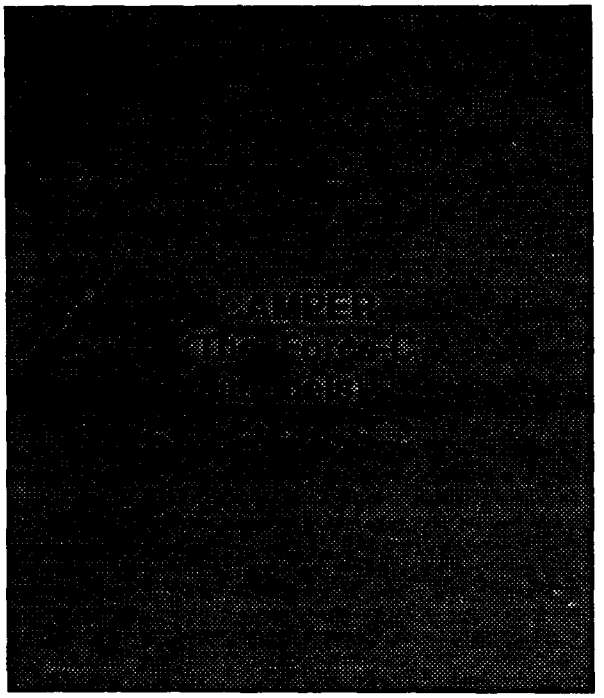
RSAS COMPOSITION BY PROGRAMMING LANGUAGE

All serious users of the RSAS must understand how the RSAS is bifurcated into parts written in the C and RAND-ABEL languages. There are two different data bases and two sets of interfaces. When a user is ready to set parameters or look at outcomes he needs to know how to do so, and this will depend upon whether the particular model of concern was written in C or RAND-ABEL. Although the distinction was once simple—that Force models were written in C and decision models in RAND-ABEL—that is no longer the case as the previous chart indicates. Indeed, all the force models for secondary theaters (those other than Central Europe and Korea) are written in RAND-ABEL using the S-Land modeling methodology (Allen and Wilson, 1987), recently renamed the CAMPAIGN-ALT methodology. The models in question are referred to as "Referee" in other documentation.

This chart shows more generally the composition of the RSAS by language. In much of what follows it will be useful to refer to Force-A vs. Force-C (the letters, of course, being for "ABEL" and "C", respectively, although Force-C also includes related data bases and software tools). Some subroutines of C programs are written in a special version of RAND-ABEL we refer to informally as "C-ABEL." Having the capability to write such subroutines makes it possible to pull out of the C code the rules and algorithms of most interest to the analyst and express them in RAND-ABEL, which is much easier to review and change. Note, however, that these subroutines use the data of the C side of the RSAS, not that of the RAND-ABEL side. Moreover, many of the RAND-ABEL-environment tools we shall describe later (e.g., the Data Editor and Interpreter) cannot be used on C-ABEL functions.

The reasons for the bifurcation in language were compelling, but many of the current difficulties in using the RSAS stem from this division, coupled with the fact that development occurred on two different coasts with different people and different technical backgrounds. In an ideal world the developments might have been more tightly coordinated, but in a world in which much was being attempted for the first time there was a premium on encouraging creativity and trying alternative solutions to overlapping problems. Now that the RSAS has stabilized we are evolving toward increasingly integrated software and a system in which the user will be only modestly aware of issues such as language compatibility. We expect other workers to have similar problems applying knowledge-based techniques to complex problems, so our experiences may prove instructive.

RSAS COMPOSITION BY SEPARATELY COMPILED PROGRAMS
(NOT SCALED BY SIZE OF PROGRAMS)



RUN.SUN
(MOST RAND-ABEL MODELS AND SUPPORTING SOFTWARE, INCLUDING FORCE-A, RED, BLUE, GREEN, INTERPRETER, AND SYSTEM MONITOR)

CONTROL PANEL/ CROSS-REFERENCE TOOL

H-TOOL

APPLICATION INTERFACE

DATA EDITOR

GRAPH TOOL

MAP TOOL

LOG TOOL

RAND-ABEL COMPILER

GRAFFER

RETARGETING TOOL

CMENT

INPUT PROCESSOR

RAND TEXT EDITOR

UNIX AND SUN UTILITIES
(e.g., Windowing System)

RSAS COMPOSITION BY SEPARATELY COMPILED PROGRAMS

It is sometimes necessary to understand how the RSAS is broken down into separate programs, although modelers and analysts should ordinarily not be concerned with such matters. Force-C's models are in one large program, CAMPER (the dark grey box on the chart). The various programs representing models written in RAND-ABEL are actually combined into a single large program named, for historical reasons, RUN.SUN (diagonally shaded box). In addition, there are numerous separately compiled utility programs making up the various interface tools. These are shown by the clear boxes, and each will be discussed later in this briefing. The diagram loosely groups them under CAMPER or RUN.SUN depending upon which they have the closest association with—however, this should not be taken to imply an exclusive association.

Some RSAC publications use the terms RSAS-C and RSAS-A to stand for CAMPER and RUN.SUN, respectively.[1] Note that they do not together constitute the entire RSAS. Another source of ambiguity is what "The RSAS" means in terms of programs. Obviously, some of the software that comes with the Sun workstation (represented by the cross-hatched box) is required to run the RSAS; however, there will be no further mention of this software here.

The RAND-ABEL Compiler and the Input Processor (shown in light grey) are both unusual in that they both are essential to the RSAS but are never used during an actual run—only during preparation. The former translates the RAND-ABEL language into the C programming language (instead of the usual assembler language to enhance RAND-ABEL's portability) and creates the initial RAND-ABEL data base. The latter compiles the Force data base.

---

[1] The diversity of terms used for RSAS components *is* confusing, and will persist until older terms go away. The appendix summarizes equivalences.

CONTENTS

CONTENTS

Let us now begin the discussion of system architecture. To do this we shall first identify the main entities in software terms, and then move on to issues of data flow and control flow.

RSAS SOFTWARE ENTITIES AND STRUCTURES

Principal Entities

- **Agents**
  - — **Force Agent to control clock and simulate physical processes (and low-level decision processes)**
  - — **Red, Blue, and Green (and Control, representing analyst)**
- **Data Bases (WSDS-C and WSDS-A)**

Background Entities

- **System Monitor as Main Program**
- **Tools for assessment, analysis, and modeling/programming**
- **Agent-to-agent communication software**
- **Language-related software**
- **Utilities**

RSAS SOFTWARE ENTITIES AND STRUCTURES

This chart distinguishes the types of entities and structures found within the RSAS. Most of the discussion will focus on the agents, data bases, and tools. There are a few items here that we should define, even though they will be discussed more later:

The WSDS-A and WSDS-C are the RAND-ABEL and C data bases, where WSDS stands for World Situation Data Set.[1]

System Monitor is the top-level system-software entity, the chief function of which is to control the sequence of the agents' execution.

---

[1]As previously mentioned, the distinction between the RAND-ABEL and C worlds is blurred, although an essential difference is which WSDS a given routine utilizes. Since they exist within two disparate pieces of software, no routine has direct access to both WSDSs.

MAJOR FEATURES OF SOFTWARE APPROACH

- Hierarchical coprocesses represent natural hierarchical objects of conceptual model
- Agents communicate through centralized World Situation Data Set (WSDS-A)
  - modularity plus efficient development of complex system
  - different "message-passing mechanism" than function calls of object-oriented programming languages
  - WSDS-A like a common blackboard, but with coded messages and a Data Dictionary for discipline and access control
- Special capabilities for WSDS saving and branching, lookaheads via "pushes" and "pops," Delta WSDSs, and extreme flexibility of user modes

MAJOR FEATURES OF THE SOFTWARE APPROACH

One of the most important software decisions was to use hierarchical coprocesses to represent the various agents. The coprocess issue is discussed more fully in the next chart.

A second important decision was to have all RAND-ABEL agents communicate through a centralized data base (the RAND-ABEL World Situation Data Set or WSDS-A). Metaphorically, these agents communicate by leaving messages on a common blackboard. However, a crucial part of the metaphor in this case is that the messages are coded so that they can be read by only the intended recipients. This is implemented through the RAND-ABEL language and its associated Data Dictionary, which include the concept of data "ownership." If one agent "owns" a certain variable, the value of that variable cannot be read "directly" by another agent.

There are instances in which it is expedient to violate modularity to make something work that is due tomorrow, or to experiment with some new concept before revising the underlying design of the model. In the RSAS we allow one agent to access the variables of another within the RAND-ABEL part of the system by providing a special syntax. For example, a Red Agent rule could include: "If *Blue's* <variable name> is <variable value> Then ..." Thus, Red would be reading Blue's internal variables. The code used for the message writing, then, is deliberately easy to break. After an exercise requiring such an expedient of module-breaking, it is easy to find and eliminate these violations of standard practice.

A third important decision made early on was to provide the capability for branching and lookaheads. In "branching" one may run a game to a certain point, save a world state (both WSDS-C and WSDS-A), continue the game, and then go back and run an excursion from the saved state. A "lookahead" is a special case of this in which an agent or human player: saves the world state (WSDS) (only part of which it knows correctly); creates one or more sets of assumptions about the opponent model, its own decisions, and other matters; runs the RSAS ahead to evaluate likely consequences; and then returns to the "real game" and makes its decisions using the lookahead simulation as a source of knowledge.

The mechanism used for implementing these concepts involves "pushes" and "pops," to use the software jargon. The metaphor here is to stack mechanisms such as that used for cafeteria trays. One creates a new world state (adds a new tray) to the top of the stack by pushing down the others. When one erases the new world state from memory after a lookahead (removes the top tray), the old one pops back up and is on top again. This can be repeated, in principle, through many levels, although we do not in fact do so because of memory constraints and conceptual complexity.

COPROCESSES

- Coprocesses: are created, stop processing to "sleep," then "awake" later to begin where they left off
- Changing object hierarchies (coprocess structure) during simulation is easy, allowing variable structure simulation
- Convenient for script-like programs representing plans
- Convenient for treating hierarchies of independent entities
- System Monitor is Main Program, charged with control
- Suitable for distributed processing; Force-C models are in a separate UNIX process

COPROCESSES

Coprocesses, an elaboration of coroutines (Knuth, 1973), are unlike ordinary functions. When a given coprocess gains control, it begins execution not from the top, but from where it left off previously. Coprocesses can, during a simulation, create (or destroy) other coprocesses, and generally execute for a period, stop processing to "sleep" while other coprocesses operate, and then wake up and proceed from where they left off. Because they can be created or destroyed readily during a simulation, they permit variable-structure simulation, which is still unusual, although there are many examples in the physical world for which this would be appropriate (Zeigler, 1986).

The implementation for the RSAS of coprocesses in UNIX (accomplished by Hall in 1984) was probably the first UNIX implementation of what we call coprocesses (coroutines had been done earlier), although subsequently other workers have independently implemented similar software under various other names such as "lightweight processes." The treatment in the RSAS, however, is probably still unique with respect to dynamic hierarchies and wakeup conditions.

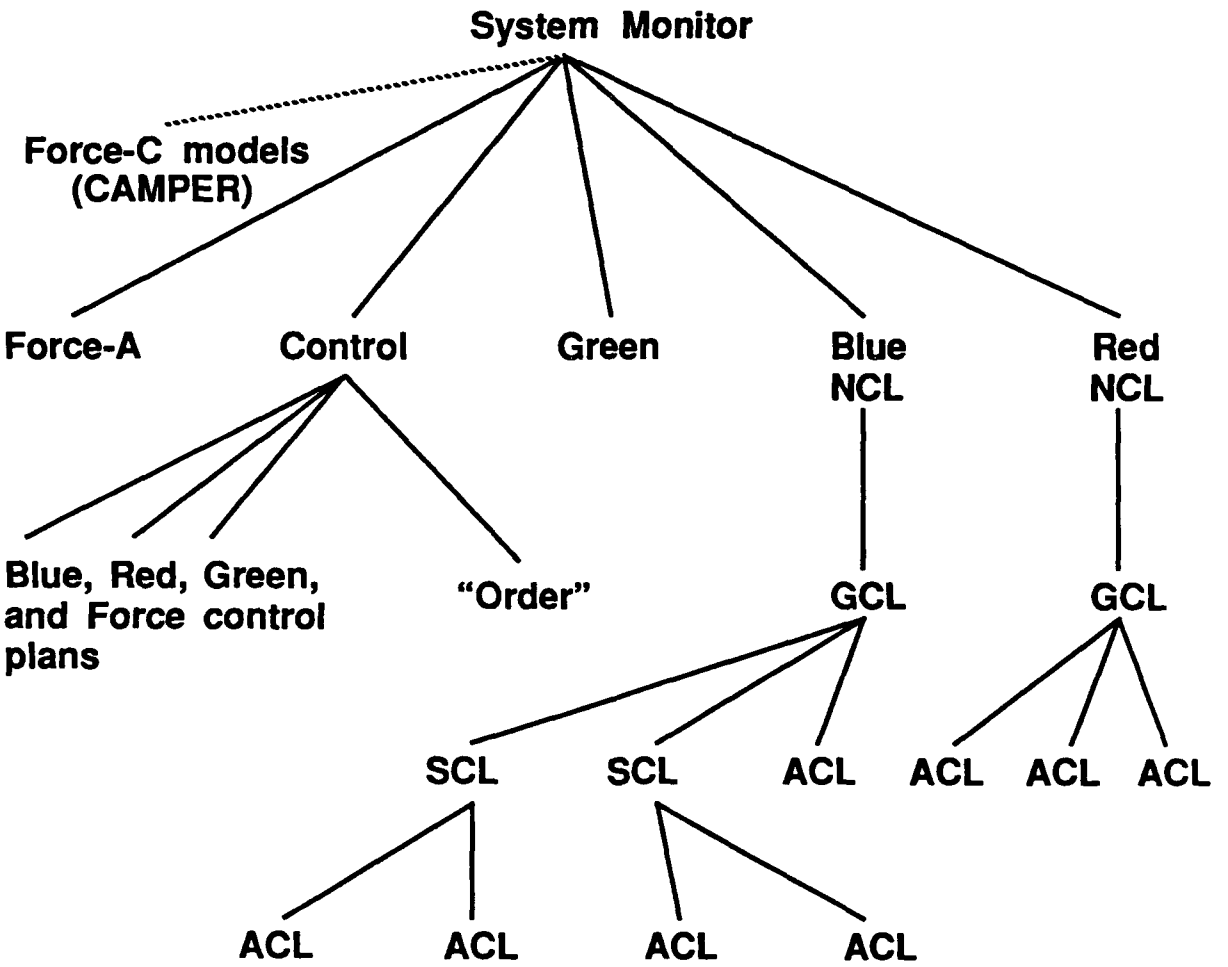
The coprocess approach was originally motivated by the opportunity it provided to represent military plans as script-like programs as suggested in early design work (Steeb and Gillogly, 1982). For example, a theater commander may order his forces alerted in one phase of operations, deployed in another, and sent into combat in yet another. In between, there are "sleep periods" for the decision models, although the simulation continues.

The coprocesses also provide an excellent mechanism for modularity. Each model entity (e.g., a given Red theater commander) has attributes and simulates processes affecting those attributes largely or completely independently of those belonging to other model entities. For example, Red's image of Blue's world and Red's decision processes in relation to Blue at the national command level do not depend directly on Blue's perceptions, or even directly on the perceptions of lower-level Red commanders. They can, of course, depend on information provided by messages from those other entities. For both substantive and software reasons, then, such modularity is very useful.

System Monitor is the "Main Program" (in the sense of C-language programs), and controls the order in which the other coprocesses execute—much as in operating systems. Each coprocess has its own wakeup rules, which Monitor tests routinely. The Force Agent is special in that it gains control whenever none of the other agents wish to awake.

Finally, let us mention that coprocesses provide an avenue toward distributed processing. As one step toward this, we have made the program containing the Force-C models into a separate UNIX process so that it can be run on a separate computer as part of a network (in a sense, then, the Force-C models are not a coprocess with the others, at least not in the usual sense). In the future, it may be possible to have Red, Blue, and Green running on separate machines with separate user interfaces, although not concurrently.

ILLUSTRATIVE "FAMILY TREE" OF RSAS COPROCESSES



Illustrative only.

(Blue and Red may have arbitrary number of SCL and ACL coprocesses; Force-A, etc., may have child coprocesses.)

..... The Force-C models (CAMPER) and System Monitor are both UNIX processes and can run on separate machines.

ILLUSTRATIVE FAMILY TREE OF RSAS COPROCESSES

The coprocess structure of the RSAS is illustrated by this chart, although it is simplified because of the bushiness of Red and Blue command-control structures. Remember that each of the entities shown can have its own wakeup rules.

Although System Monitor is the main program controlling all of the others, both it and the Force-C models (also called CAMPER) are UNIX processes capable of running independently. Within RAND, CAMPER and the decision models are usually run on one Sun workstation while the rest of the RSAS (i.e., the interface tools) is run on another. This first step toward distributed processing proved efficient and informative. In the future, distributed processing will become more common and, for example, the Red Agent may operate on a different workstation than the Blue Agent.

There are some entities noted here that we shall discuss later in more detail. These are special coprocesses subordinate to the Control Agent, all of which give the analyst convenient ways to schedule interventions without having to watch the game in detail, stop it, and make changes interactively.

AN ASIDE: TIES TO OBJECT-ORIENTED PROGRAMMING

- Features of standard object-oriented programming languages (OOP) (e.g. SMALLTALK, ROSS, ...):
  - Data hiding
  - Inheritance
  - Message-passing
- Problems with current OOP languages:
  - Not suitable when objects interact too much (e.g., continuously) or too intricately
  - Clumsy in treating interactions with "fields"
  - Based on single-paradigm approaches
- RSAS features:
  - Object-oriented modeling for agent architecture
  - Modularity (data hiding) through language features of ownership
  - No built-in inheritance features
  - Object-to-object communications through a central data base, not function calls

AN ASIDE: TIES TO OBJECT-ORIENTED PROGRAMMING

We are often asked whether the RSAS uses object-oriented programming. The answer is complicated because it is not entirely clear what object-oriented programming (OOP) is, or at least what it will be in the future.

Historically, "object-oriented programming" has often been used synonymously with programming in certain specialized languages such as SMALLTALK-80 (Goldberg and Kay, 1976), ROSS (McArthur and Klahr, 1982), or the system discussed in Reddy (1987) and in earlier work by Reddy and Fox. These typically have features called *data hiding, inheritance, and message-passing*. Each of these features, in turn, is a particular way of addressing particular issues (i.e., modularity, the hierarchical relationships among variables and operations on them, and the mechanism by which different software modules interact with each other). Roughly speaking, the idea is to build programs from entities conceptually like natural objects and the relationships existing between them. These objects have attributes and processes affecting those attributes ("methods") that are known only to themselves. Objects interact by sending messages to each other—similar to procedure calls in other languages—and (in classic OOP) share data in no other way. There are various classes of objects, with each *instance* (i.e. individual object) of a class having identical methods but data entirely private to itself (a form of data hiding). Hierarchies of classes exist, with related classes sharing methods but adding (or modifying) some of their own (such sharing is termed *inheritance*).

Unfortunately, despite current enthusiasms for them, OOP methods are no panacea—indeed, they are often highly inappropriate. The OOP methods work well for systems of discrete entities that are relatively independent or have strictly defined interrelationships. Current OOP methods are clumsy for systems of entities interacting continuously with the environment (e.g., gravity or terrain) or with one another in complex ways. There are analogies here to physics, in which it is well known that the appropriate representation of a system is sometimes particle-oriented, field-oriented, or a hybrid. At the time the RSAS was designed there were no well-developed high-level languages that would allow us to mix object-oriented and "field-oriented" or procedure-oriented representations conveniently. Even those newer languages that allow such mixtures of techniques generally do so at the expense of one paradigm or the other, or as an ill-fitting hybrid.

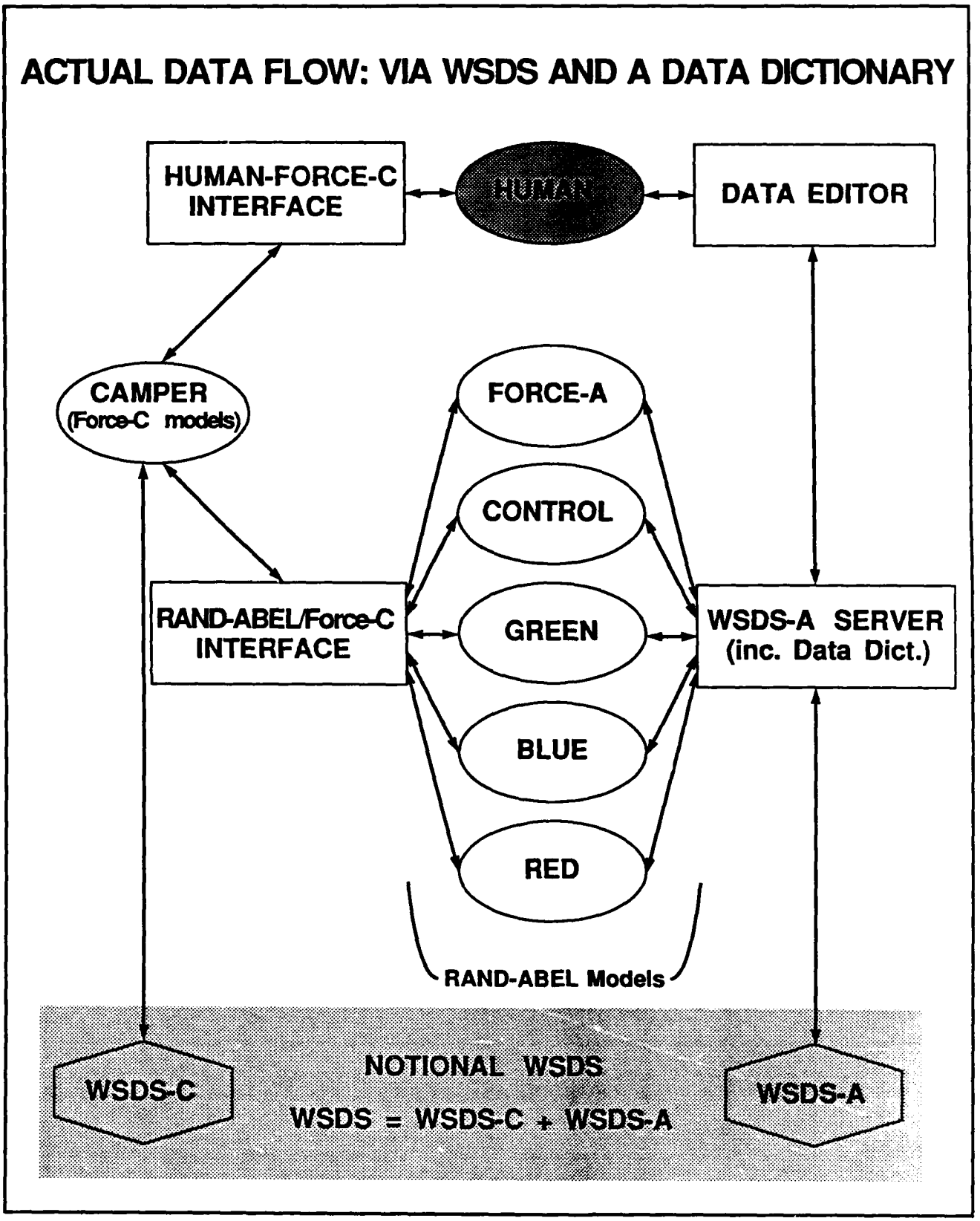
Another problem with current OOP languages is that their implementation of inheritance, motivated by the search-oriented inference techniques of artificial intelligence research, are extremely inefficient by comparison with what can be accomplished with more specific control structure and the use of pointers. Efficiency is a major issue for systems like

the RSAS. Some newer languages—for example, C++—reach a more efficient compromise but are ill-suited to the highly symbolic nature of political-military simulation.

With this background, then, how does the RSAS relate or not relate to OOP methods? It achieves what amounts to data-hiding modularity through the use of coprocesses and the ownership features of the RAND-ABEL language. It does not have built-in inheritance features, although it treats related issues through explicit programming; we have no plans to incorporate the inefficient search-oriented inheritance features of classic OOP languages. Agents do send messages among each other in the RSAS, but the message-passing mechanism is implemented in a centralized data base using semaphores and protected data areas (not the function-call mechanism of OOP languages).

The Force-C simulation would be regarded by some as a natural candidate for object-oriented programming. The number of entities and interactions is so great, however, and the role of continuous processes so prevalent, that a classic OOP approach would have proven extremely inefficient and, probably, clumsy by comparison with the pointer-exploiting methods we actually employed. At the same time, the resulting Force-C source code (and to some extent the code for supporting software) is very complex and will doubtless not be easy to maintain. It would have been desirable to have had a variant of C with at least some of the object-oriented features built in (primarily data-hiding). Although such variants of C exist today, they were not available when we began and we have no direct experience with them as yet.

In summary, the RSAS embodies many of the very complications that limit the usefulness of the original object-oriented programming languages. Nonetheless, it is functionally object-oriented in its top-level architecture and it incorporates what amounts to data hiding and a loose form of "message-passing." Although the RSAS source code deals with many inheritance-related issues, it does not do so using inheritance features built into the language itself. Although there may be some areas where an inheritance mechanism would have aided the programming process, in many cases the complex interrelationships and exceptions would have made such mechanisms extremely clumsy.


ACTUAL DATA FLOW: VIA WSDS AND A DATA DICTIONARY
HUMAN-FORCE-C INTERFACE
HUMAN
DATA EDITOR
CAMPER (Force-C models)
FORCE-A
CONTROL
GREEN
BLUE
RED
RAND-ABEL/Force-C INTERFACE
WSDS-A SERVER (inc. Data Dict.)
RAND-ABEL Models
WSDS-C
NOTIONAL WSDS
WSDS = WSDS-C + WSDS-A
WSDS-A

ACTUAL DATA FLOW: VIA WSDS AND A DATA DICTIONARY

Returning to the main thrusts of the briefing, this chart anticipates the discussion of data flow that follows by showing at a top level how the various entities are related in a data flow diagram. It is important to observe the distinction between the software implementation of data flow and the apparent data flow suggested by the earlier charts on the conceptual model. In particular, where the conceptual model shows agents sending each other messages, this chart demonstrates that the messages do not go directly from one agent to another, but rather into a centralized data base where certain other agents can read them. Also, while we will often refer to "the WSDS," this chart reminds us that there are two data bases, the WSDS-A and the WSDS-C; "the WSDS" is an abstraction.

This chart also shows a WSDS-A "server," which includes a Data Dictionary. This is an important software entity allowing interface software as well as the agents to access WSDS-A data without requiring the agents to understand details of WSDS-A structures. That is, interface software can merely refer to variable names without having to worry about storage locations in memory, locations that could change as variables are added or subtracted from the system. When an analyst introduces a RAND-ABEL variable, he must declare it in the Data Dictionary, which includes information on the variable's type, ownership (access restrictions), and semantic meaning. There exists no comparable server for the C side of the RSAS.

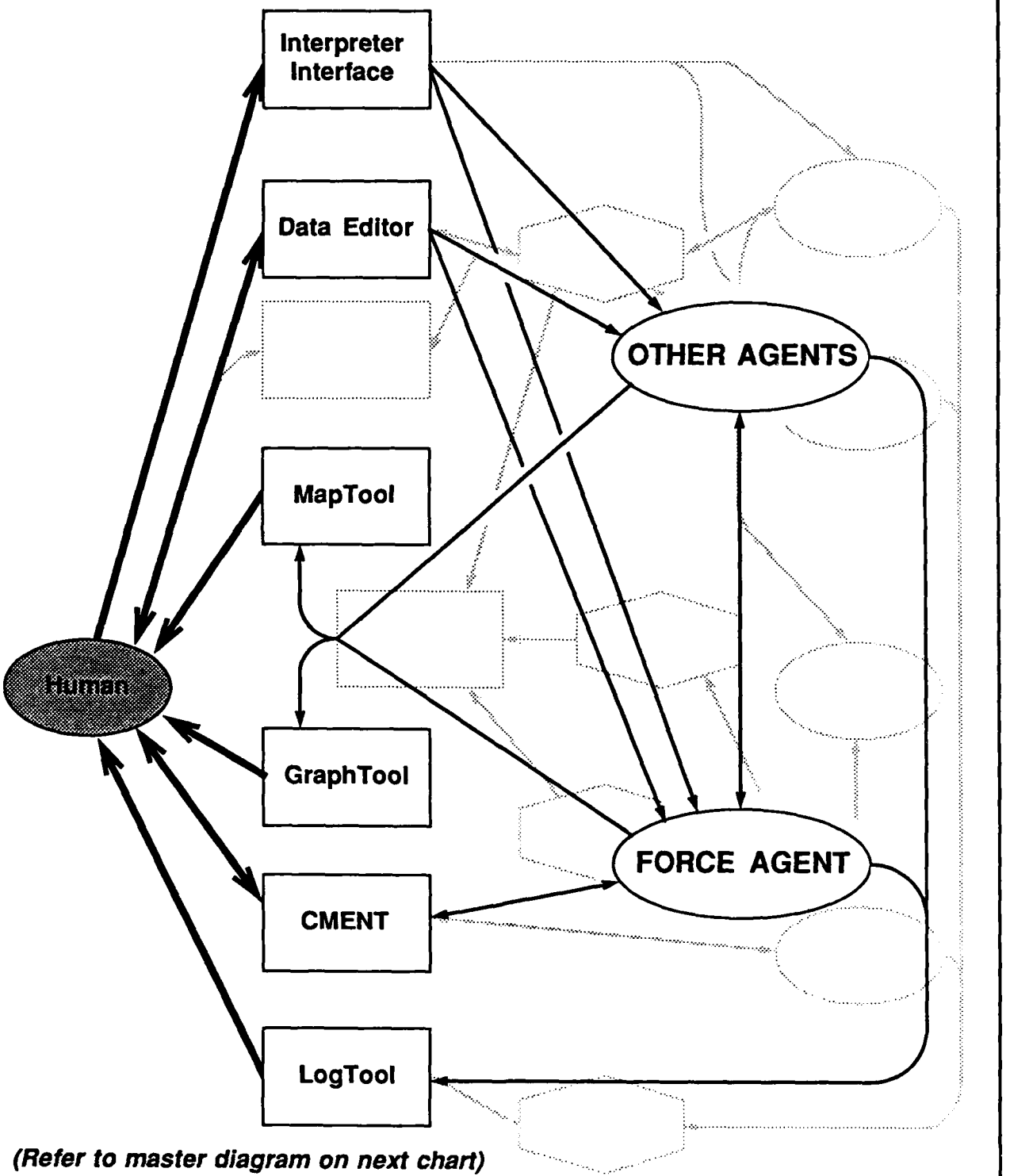
Finally, let us emphasize a terminological problem that continues to haunt RSAS users. Most long-term RSAS developers and users refer to the Force-C models or CAMPER as "Force," because originally they were identical as mentioned earlier (i.e., there was no Force-A). This can create serious misunderstandings, however, when one is discussing either interface issues or higher-level modeling issues. (See also the appendix.)

CONTENTS

DATA FLOW IN THE RSAS

One of the best ways to characterize the high-level structure of software is with data flow diagrams, because these describe the way information flows without worrying about aspects of control flow that are often arbitrary or implementation-specific. We shall now examine a number of data flow diagrams to introduce and describe major interfaces.

PRINCIPAL HUMAN INTERFACES FOR DATA
(FUNCTIONAL OR MODEL-ORIENTED VIEW)

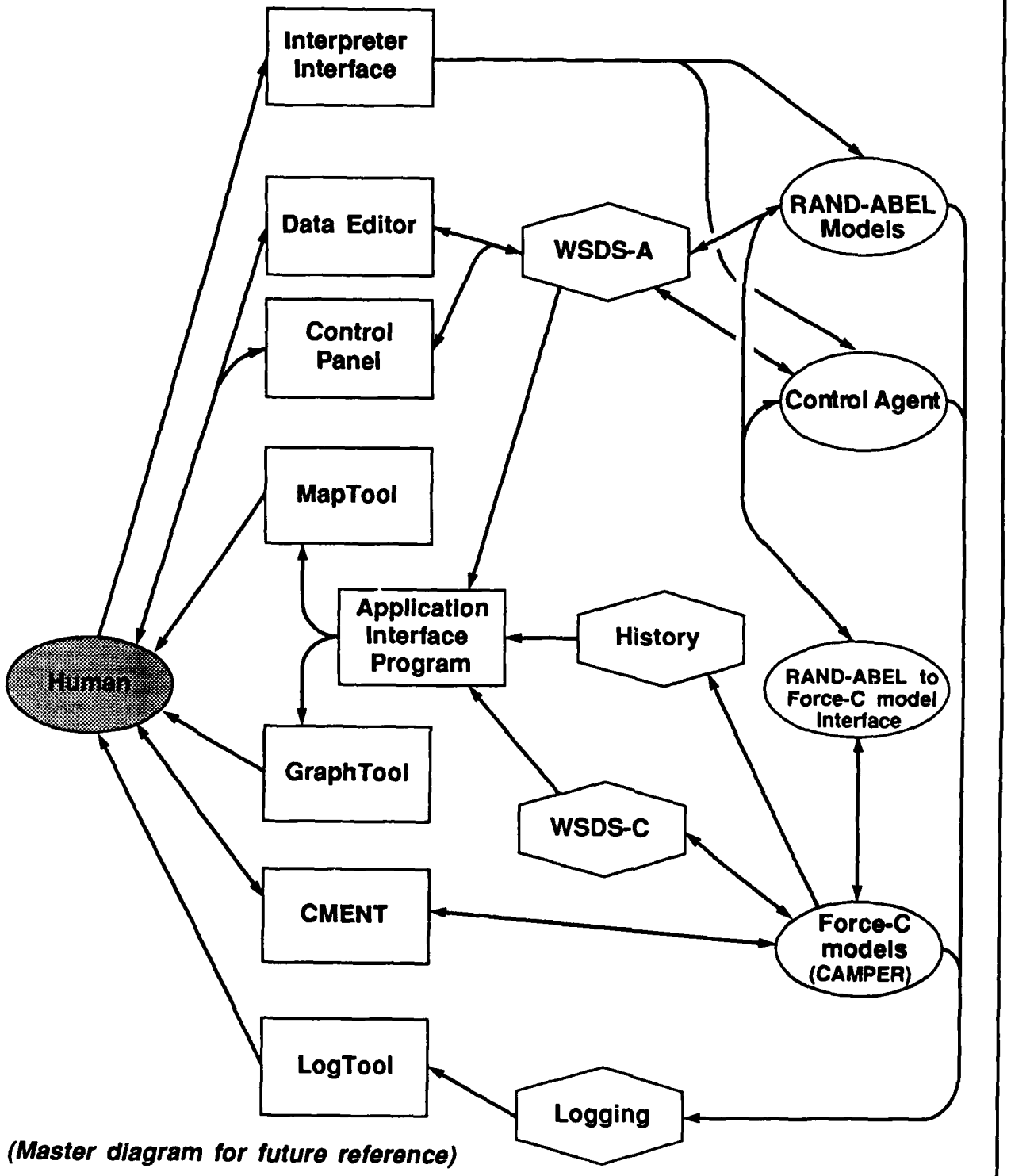


(Refer to master diagram on next chart)

PRINCIPAL HUMAN INTERFACES FOR DATA

This chart identifies the most important RSAS man-machine interfaces, of which there are six that are especially important and conceptually different. We shall show some others later, notably CONTROL PANEL (shown in outline), which merely provides a simple top-level interface to System Monitor, GRAFFER (not shown here, even in outline), which is an essential graphics interface for certain types of analysis but is being phased out as its functions are taken over by more general graphics tools, and the RAND text-editing program, "E," used in manipulating source code, databases, display formats, etc., and in reading log files.

DATA FLOW AND INTERFACES IN RSAS
(Master Diagram)



(Master diagram for future reference)

DATA FLOW AND INTERFACES IN RSAS (AGGREGATED)

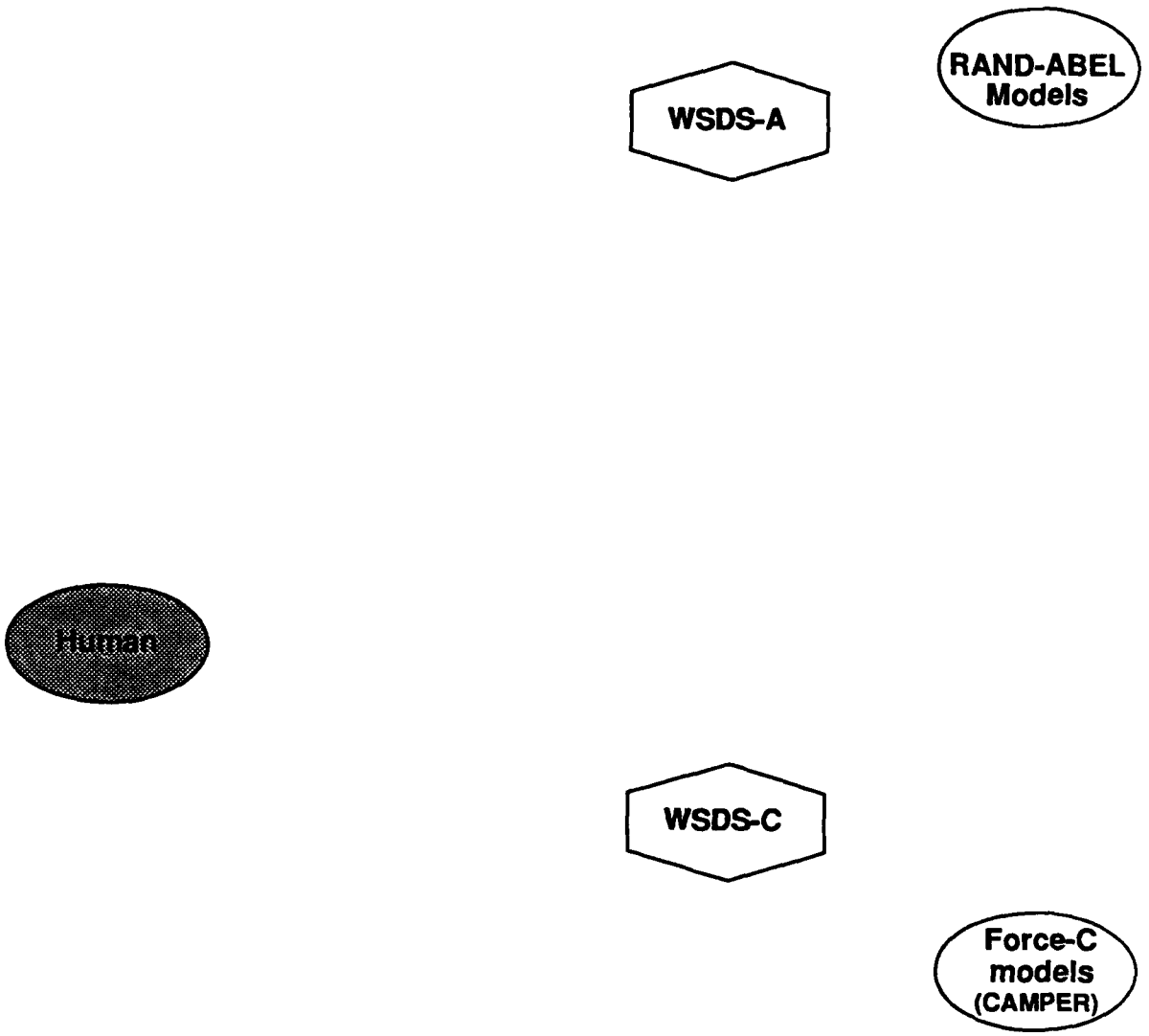
Diagrams such as this can be extremely useful, but they can also be intimidating when first viewed. The most important points to notice probably are these:

- The C-language entities Force-C models (CAMPER), WSDS-C, and CMENT are on the bottom; the RAND-ABEL entities are on the top (see also the next chart).
- There are communications between the two classes of entity, but this is complex enough to merit showing explicitly a RAND-ABEL/Force-C Model Interface, a complex piece of software developed by colleagues Robert Weissler and Barry Wilson. Remember that at the level of software the RAND-ABEL and C programs live in two different worlds, and can communicate only when special arrangements have been made. This would be trivial if only a few variables were at issue, but it is not at all trivial in a large and complex system exploiting pointers and functions, and being worked upon by many people, both analysts and developers.[1]
- All the interface tools, with the exception of the logging tools, communicate using a common "Tool Interchange Language," or TIL. TIL is designed to be efficient and robust (it depends on streams of ASCII data), but easy to debug by virtue of being directly readable by the programmer. In fact, several system configuration files also use the TIL syntax. This uniform approach has greatly aided our efforts. The TIL language is described in on-line RSAS documentation under the Data Editor.
- The graphics and log interfaces currently are used strictly for output. They provide information both during a game and afterward. The other interfaces are for input, or for both input and output.
- The Data Editor and CMENT windows perform similar functions for the RAND-ABEL and C worlds, respectively, providing for the interactive viewing and changing of data. However, CMENT is really just a menu- and window-oriented front-end for the existing command-line interface contained in Force-C, while the Data Editor (plus the Interpreter) form the basic interface to the RAND-ABEL models.
- The Interpreter interface is different in kind because here the user is not merely changing parameters of the models but the models themselves. That is, he may wish to review and then change the underlying rules (or parameter values) in the middle of an RSAS simulation.

---

[1]The basic interface emphasized minimizing the number of human interactions necessary to coordinate changes in source code connecting the RAND-ABEL and C programs. Any model on the "RAND-ABEL side" can obtain data from the "C side" by calling and parsing in a routinized way the data stream constituting the human-readable data displays developed previously for users of the Force-C combat models. So long as the data needed exists in such a display (and it usually does, because if it is important enough to be needed by a RAND-ABEL model it is important enough to be in a human-readable display), obtaining the data can be accomplished by a RAND-ABEL-side programmer without consulting (and disrupting the work of) a C-side programmer. When tighter coordination is possible a more efficient method can be used involving "dumps" of unencoded data rather than displays. Versions after 3.5 will do most of their intercommunication in this way.

OBJECTS OF DATA FLOW AND INTERFACES

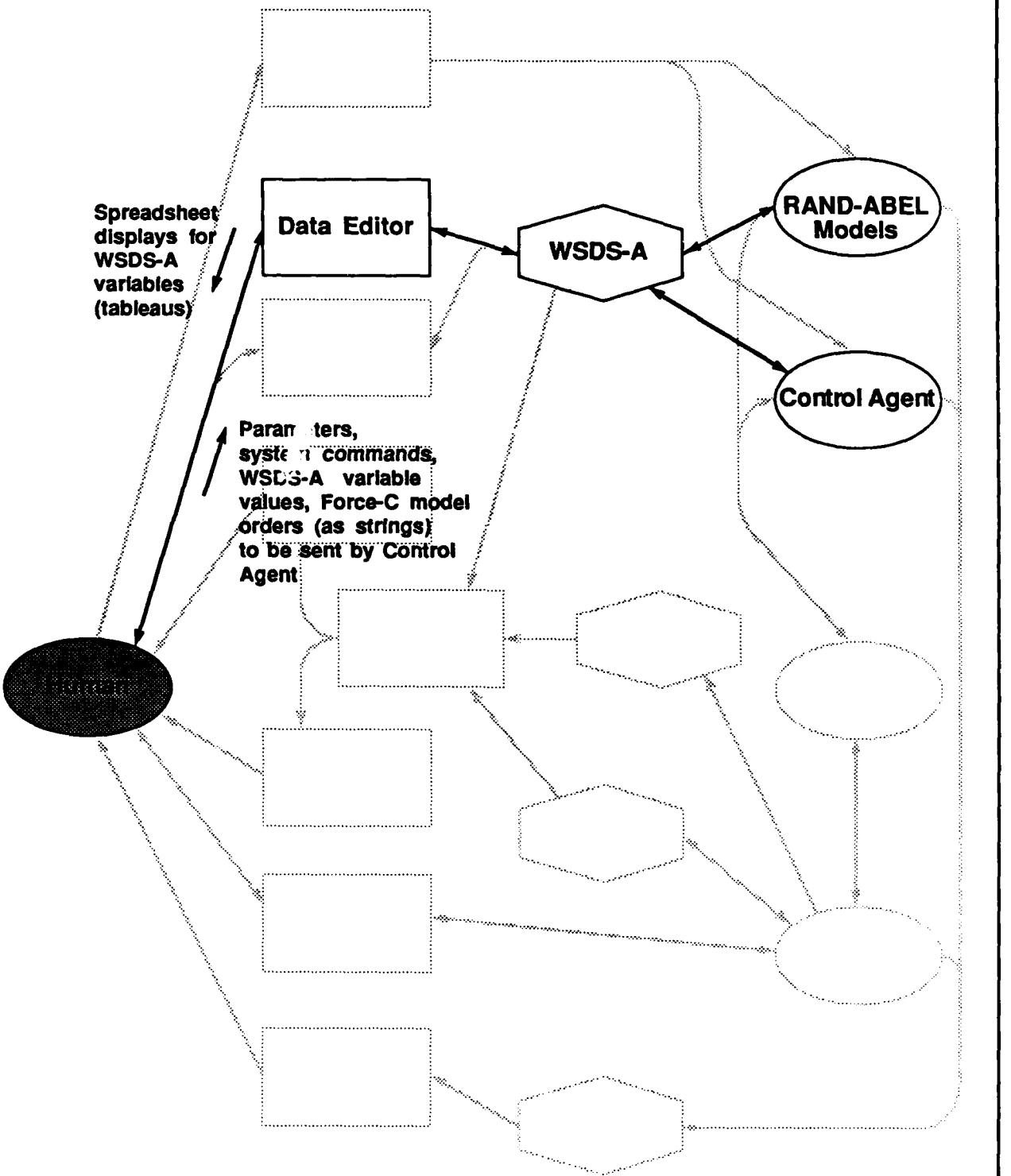


OBJECTS OF DATA FLOW AND INTERFACES

(Already discussed)

INTERACTING WITH THE RSAS DATA EDITOR

(A Full-screen Interactive Editor)



INTERACTING WITH THE RSAS DATA EDITOR

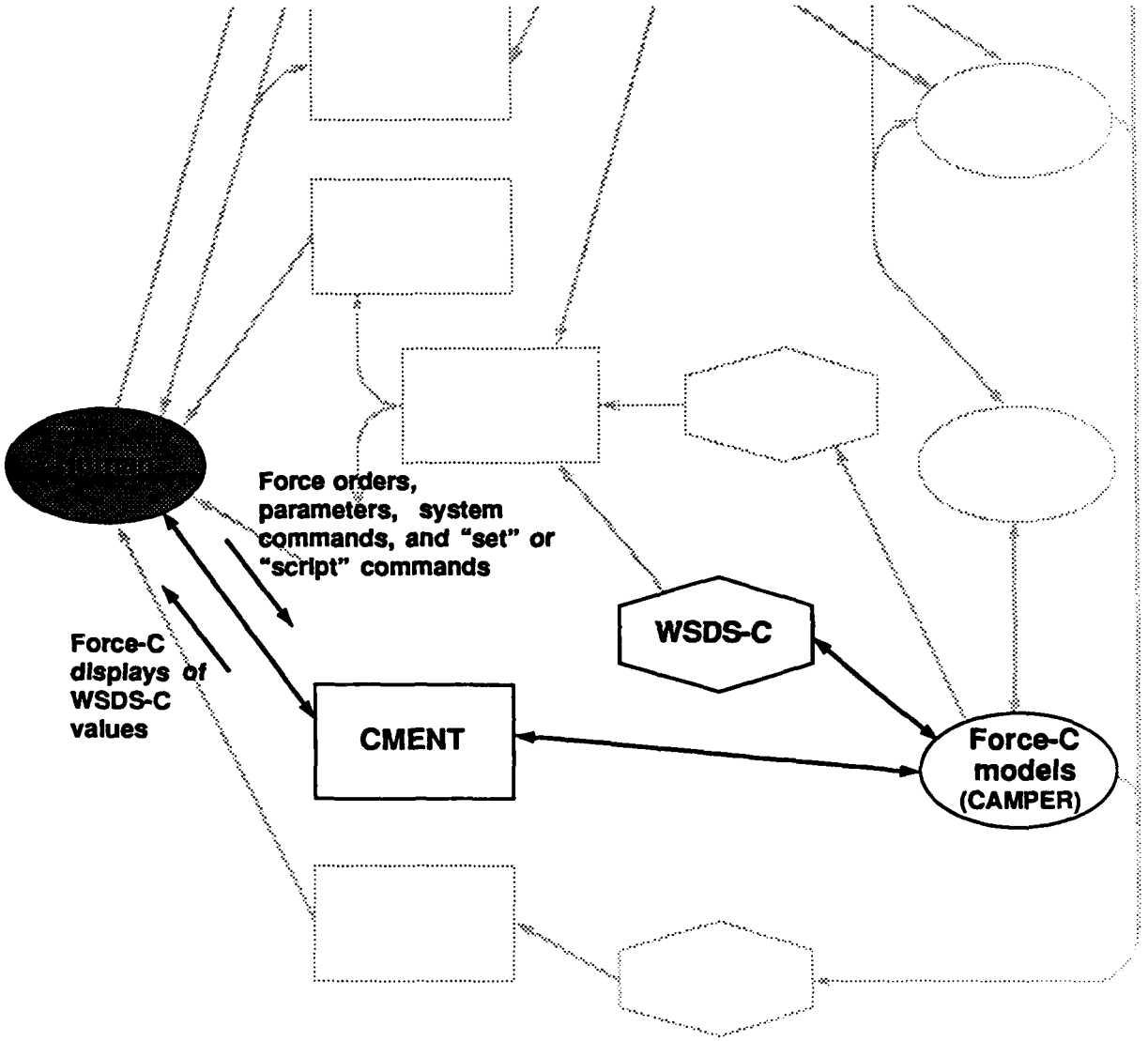
Let us now begin talking through the parts of the RSAS rather than trying to comprehend all the relationships simultaneously. This chart focuses on the principal man-machine interface to the RAND-ABEL programs, the Data Editor. The Data Editor is a spreadsheet-like interface in which one uses a mouse and menus to observe, select, and modify data (see references by colleague Jean LaCasse). Current values of WSDS-A data are displayed in a tabular form chosen from a large selection of layouts called "tableaus"; the values displayed can be modified interactively by selecting from a list of possible values or, when appropriate, by typing in a value. When cleverly designed, the tableaus can summarize all the parameters relevant to a particular issue, and can include commentary to help the analyst make wise choices and avoid common blunders.

The ability to create a great variety of data displays makes the Data Editor especially valuable to analysts, since they can use a text editor and "tableau definition files" to create tableaus displaying exactly the data needed for particular studies. Conventional interfaces require a professional programmer for such tailoring, which is usually required since the original programmers of an interface rarely anticipate all the needs of future users.

One matter with which the user must be concerned is whether changing a given variable will actually have the desired effect on the simulation. If the models have been designed appropriately, the user may choose to override certain model decisions and then "lock the changes in." Otherwise, the user might wind up setting values that the model itself would likely override at some later point in the simulation. Thus. in using the Data Editor it is important to understand which variables may actually be changed meaningfully.

The Data Editor can also be used to set variables in the Force-C models. Some WSDS-A variables amount to instructions as to when Control Agent should send Force-C specific orders.

INTERACTING WITH FORCE AGENT VIA CMENT



INTERACTING WITH FORCE AGENT VIA CMENT

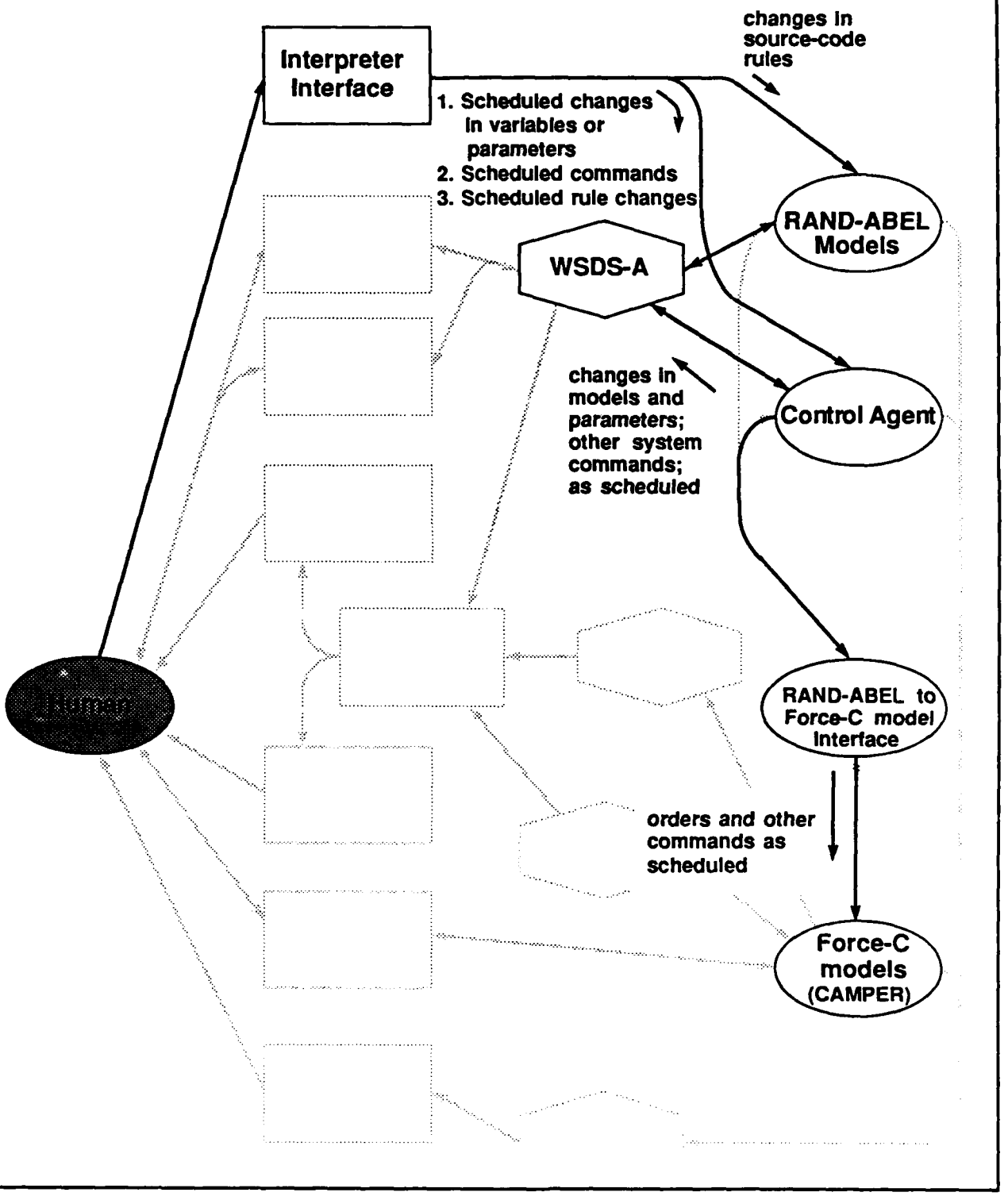
From a user's viewpoint, the CMENT window is the primary interface with the C side. Developed by colleague Mark Hoyer, CMENT includes nested menus and some interactive tableaus, which are similar in purpose to those of the Data Editor.[1]

Originally, the Force interface was a line-by-line command interface (e.g., the user would type parameter-setting or order-sending commands such as "set landwar US intens 0.6" (a fictitious example), and then be queried for additional information one line at a time). This interface with syntactically well-defined orders remains the comprehensive and definitive interface, but the user of CMENT may either enter such orders or exploit the various menus and tableaus. As the system matures, CMENT's nested menus and interactive tableaus are becoming increasingly important—not just because they reduce typing, but because they provide a better sense for the models being used. CMENT displays and menus can be changed by editing its configuration file, but must follow closely the framework defined by Force-C's underlying command-line interface.

---

[1]Optionally, users may employ the "F-Window," which provides only the line-by-line command interface without the CMENT-specific options allowing the uses of the mouse on nested menus. Since CMENT uses a fairly large amount of memory, those users who prefer the bare command interface and who do not need CMENT's global displays often use F-Window instead.

THE RAND-ABEL INTERPRETER INTERFACE



THE RAND-ABEL INTERPRETER INTERFACE

Having discussed briefly the two principal interfaces with data, let us now discuss the interpreter interface, which allows us to change RAND-ABEL *source code* during an RSAS simulation—rather than quitting, recompiling, and starting over again. (This does not apply to RAND-ABEL code within the Force-C models.)

RAND-ABEL is not interpretive in a line-by-line fashion as in, say, BASIC; in the RSAS one must make changes to a complete function. The user copies the function of interest to a special file in the INT directory and edits the function as desired. (This involves a word-processing editor developed by RAND called "E", which is not discussed in this Note even though it is, in a sense, one of the system-software interfaces.) When execution of the RSAS is continued, the revised function will be executed rather than the originally compiled function. Further changes can be made by repeating this procedure.

As this chart illustrates, the human analyst can use the interpreter to make changes "now" to Red, Blue, or Green Agents (or to the Force-A models); alternatively, he can use interpreted functions within Control Agent to *schedule* changes in terms of time or conditions. As a special case, Control Agent can be instructed to make the changes before the rest of the simulation proceeds, setting new values in the WSDS-A or sending parameter values to the Force-C models.

CONTROL AGENT

TECHNICAL CHARACTER: Same as Red, Blue, or Green Agents

FUNCTIONS: to act for the user in doing things the user would otherwise do interactively

EXAMPLES OF ACTIONS SCHEDULED IN CONTROL AGENT:

- **Set system parameters (issue system commands) affecting logs, displays, game duration, and choice of models**
- **Adjust model inputs by changing parameters**
- **Adjust model outputs**
- **Supplement model outputs (e.g., issue additional force orders on behalf of Red, Blue, or Green)**
- **Supplement or change RAND-ABEL model functions (i.e., add rules, including algorithms)**

CONTROL AGENT

The next four charts form a group defining the function and character of the Control Agent, which is technically similar to the Red, Blue, and Green Agents in being a coprocess with its own wakeup rules and, if desired, phases, moves, and sleeps. The Control Agent acts for the analyst, who otherwise might have to sit and watch the simulation proceed so he could intervene at appropriate times to change parameters, override decisions, change rules, or whatever. If the analyst can figure out what he wants to do in advance, he can schedule those interventions through the good offices of the Control Agent. Some examples may be useful:

1. Set system parameters such as log levels or the parameter specifying which NCL model is to be employed (e.g., SamN vs. SamO).
2. Adjust model inputs (e.g., French temperament or kills per sortie by aircraft).
3. Adjust model outputs (e.g., French cooperation).
4. Supplement model outputs (e.g., issue additional force orders on behalf of Red, Blue, or Green—orders not specified in the basic models).
5. Change particular model functions (e.g., a particular rule or algorithm), perhaps only temporarily or only for a particular run.

A confusing aspect of this is that analysts often want to change Red, Blue, Green, and Force models on the margin without disrupting the complex structure of the agents as a whole. If the analyst were doing this interactively, it would mean he would override certain decisions, or perhaps change the rules in just one or two highly selected areas and only under certain circumstances. By putting these interventions in Control Agent the analyst is, in a sense, dividing the content of the decision models (e.g., in a given simulation part of the Red Agent will be "in Red"—i.e., in Red Agent files—while a part will be "in Control Agent"), but this is precisely what he often wants to do in order to keep all his assumptions in one place. Thus the instructions in Control Agent are essentially exceptions to the basic defaults.

CONTROL AGENT MODES

SCENARIO GENERATOR (User-Generated or Scripted Mode)

- User schedules interventions with Data Editor, selecting from menus
- Control Agent passes instructions to relevant agents at appropriate times

CONTROL-PLAN FUNCTIONS (for Red, Blue, Green, and Force)

- User schedules interventions in RAND-ABEL plan, analogous to analytic war plans with sleeps, moves, wakeups, etc.
- Control Plan could be compiled, but is usually interpreted
- Maximizes flexibility and collects interventions in one place, but requires some ability to read and change RAND-ABEL
- Can substitute for or supplement Scenario-Generator inputs

ORDER FUNCTION

- User inserts instructions in special function and interprets
- User selects "Order" on H-Tool display
- Instructions are executed immediately, and only once

CONTROL AGENT MODES

There are three modes for using Control Agent: the Scenario Generator, Control Plans, and Order Functions. The first involves filling out Data Editor spreadsheets to schedule interventions. This *Scenario Generator* method developed by Robert Weissler is simple and convenient, and does not require knowing variable names or values. However, it limits flexibility and may prove less convenient for analysis than the second approach. We will show more detail on this later.

The Control Plan approach is one in which the user writes a simple RAND-ABEL plan scheduling the interventions he wants Control Agent to make. Usually he will do this by copying one or more functions from an existing plan (e.g., the function for deployment of forces to a particular theater) and changing it on the margin to represent his own assumptions. With programmer assistance (at least the first time through) this can be very straightforward and efficient.

Sometimes the user will need to be sure that his instructions are executed precisely at a given time or when a given condition is satisfied. This may not be a simple matter because the instructions are parts of functions that ordinarily would not be executed until the appropriate agent wakes up and decides to execute those functions as part of its (possibly complex) basic control structure. To assure immediate execution (e.g. "I want to deploy those forces now, which I forgot to allow for when I developed the original analytic war plan") there is a special "order function" developed by Barry Wilson and William Schwabe.

All of these options are documented on-line in the INT directory of the RSAS.

COMPONENTS OF AN ANALYST-PLAN FILE

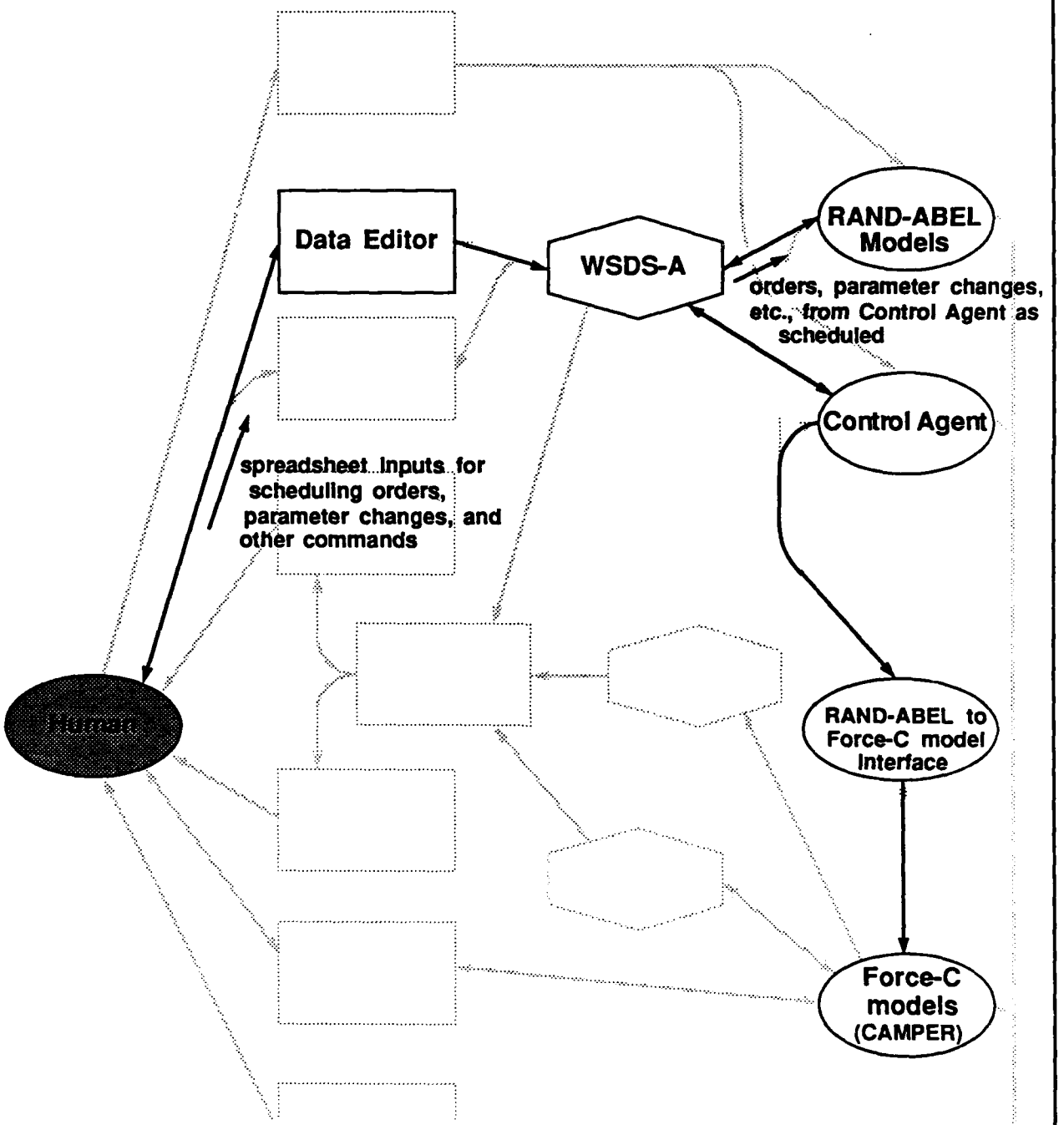
Analyst-Plan File	
Owner: Red. **Define** <Red-function-name>: <various statements> **End.**	*Change in rules: executes as part of Red coprocess*
Owner: Blue. **Define** <Blue-function-name>: <various statements> **End.**	*Change in rules: executes as part of Blue coprocess*
Owner: Red. **Define Control-plan:** <various statements> **End.**	*Analyst overriding or adding to Red outputs: executes as part of Control coprocess*
Owner: Blue. **Define Control-plan:** <various statements> **End.**	*Analyst overriding or adding to Blue outputs: executes as part of Control coprocess*

COMPONENTS OF AN ANALYST-PLAN FILE

The concept of an analyst plan has not been described in previous RSAS documentation, but is proving very important in practical day-to-day work. An analyst wants to tightly control the variables affecting the simulation. In particular, analysts often want to state all their assumptions "in one place" where they can be viewed at a glance. In practice these assumptions represent changes from some baseline; in the RSAS this corresponds to having a baseline consisting of Red, Blue, Green, and Force Agents with the analyst's assumptions consisting of selective changes in rules or parameter values. The analyst-plan file contains RAND-ABEL code specifying these changes.

This chart shows an analyst-plan file schematically. A more general depiction would include segments for Green and Force Agent changes. Note that in this example the analyst-plan file includes changes to the Red and Blue Agent models (i.e., rule changes) that apply immediately once the game begins. It also includes two control plans of the type discussed in earlier charts. These schedule additional changes that can be made by adjusting Red or Blue parameters during the game.

DATA FLOW FOR USING THE SCENARIO GENERATOR MECHANISM TO SCHEDULE INPUTS
(USER-GENERATED OR SCRIPTED MODE)



DATA FLOW FOR THE SCENARIO GENERATOR MECHANISM

As suggested earlier there are several ways to use Control Agent. The analyst-plan file is especially convenient, but requires working with computer code—although it is RAND-ABEL, which is easier to work with than most computer languages. An alternative is to use the Scenario Generator mechanism depicted in this chart. With this, the user merely fills out a spreadsheet using a mouse and menus. Underlying software then translates this into the same type of instructions that could have been introduced directly by writing some RAND-ABEL code as described in the preceding charts.

The Scenario Generator mechanism is especially convenient for demonstrations, for getting users acquainted with the system at an elementary level (having to change computer code can be intimidating), and for preparing a variety of baseline world states. It is less convenient for entering the changes between one excursion and another because the variables may appear in a large number of different Data Editor tableaus. It is possible to design one's own single-page tableau tailored to a particular study, but someone who has achieved enough competence to do that will probably feel more comfortable working with analyst-plan files instead because, in our experience at least, they will be eager to go through numerous cases and will be impatient with the easy-to-use but time-consuming mouse-and-menu approach. It is generally faster to type the instructions into a file.

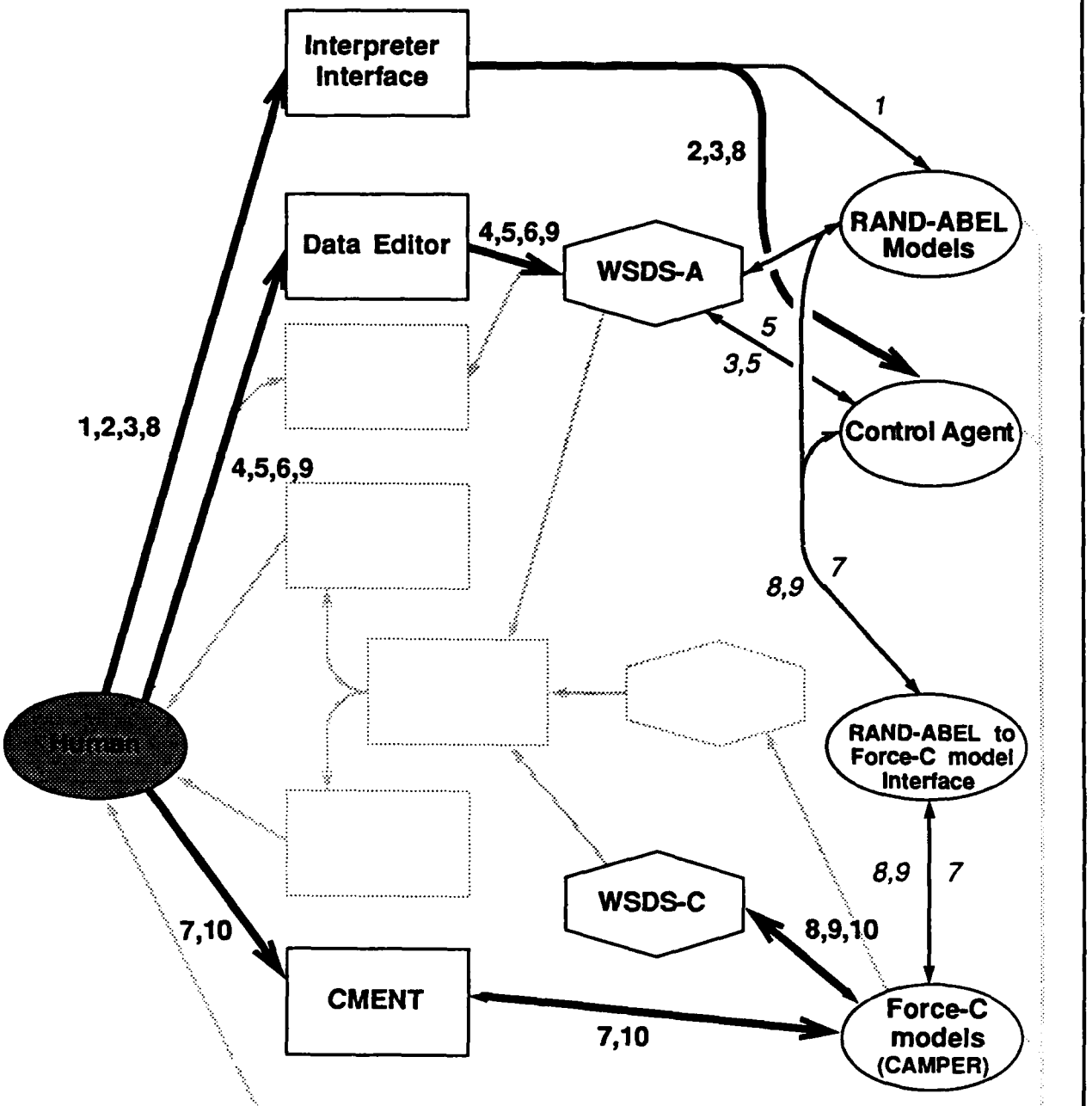
The most important point to make here is that the Scenario Generator mechanism is relatively user-friendly, but the options it provides are only those that have been anticipated. In working with the RSAS users frequently want to make changes that are easy to do in RAND-ABEL but that cannot be made using existing Scenario-Generator tableaus.

DETAILS OF HUMAN INPUTS TO FORCE AGENT

(Rule changes, parameters, orders, and other commands)

[excludes time-zero data bases]

SANDTABLING MODE



SEE FOLLOWING CHART FOR LEGEND

DETAILS OF HUMAN INPUTS TO FORCE AGENT

This and the next three charts describe paths by which a user may specify inputs to the Force Agent models. We shall not discuss them in detail here, but they may prove useful as reference—e.g., in clarifying how and why different users are going through such different operations to do "the same thing," and in suggesting when one wants to use one method rather than another for making changes in the force model parameters. The numbers appearing along the arrows are keyed to the third chart, NOTES ON FORCE AGENT INTERFACES. The first chart highlights certain paths appropriate to users doing *sandtabling* in which the user is *exploring*—trying this and that but not worrying about tight control over the variables or careful record keeping. Everyone must start with the sandtabling, because it is by using the CMENT window and the Data Editor that one gets to understand the models and variables, as well as to learn their precise names. Only when these things have been learned can one go on to the *analysis* stage in which one wants very tight control and very good and compact records. Here most analysts prefer to use analyst-plan and scenario-generator mechanisms almost exclusively.[1]

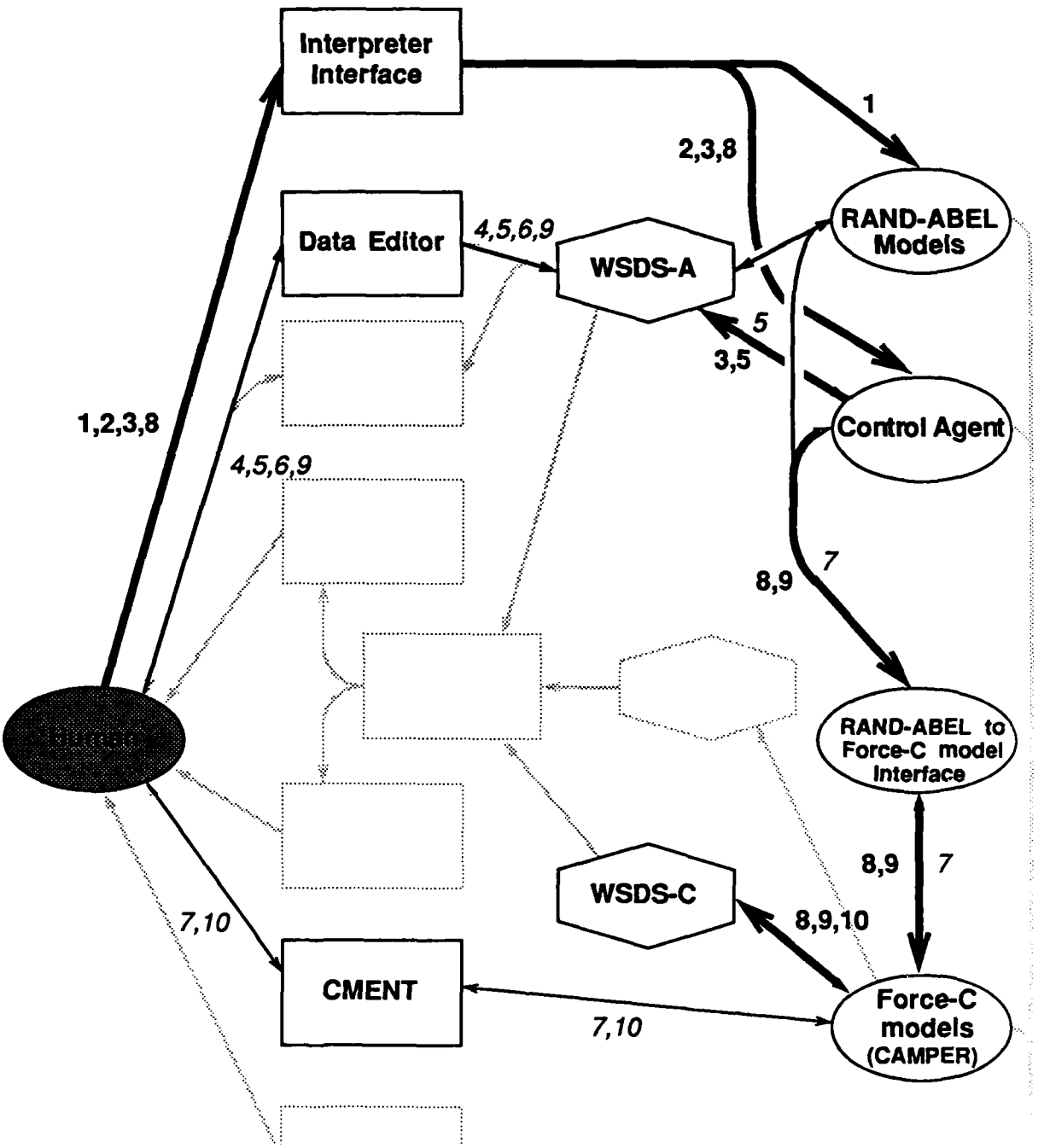
---

[1]We should mention here that in some applications of the RSAS focused on the European Central Region or strategic nuclear forces, analysts sometimes work exclusively with Force-C (CAMPER with its associated database and tools). In these instances they construct something called "Use Files"—essentially scripts of commands to the Force-C models—to define their various cases. They play the same role as Analyst Plans but are much less flexible, since they do not contain conditional logic involving model variables or exploit libraries of analytic war plans. Use Files developed in such work are often used to construct the first-cut versions of analytic war plans after a study is concluded. Thereafter, adaptive logic can be added so that the plans preserve the original thinking while adding flexibility.

DETAILS OF HUMAN INPUTS TO FORCE AGENT

[EXCLUDES TIME-ZERO DATA BASES]

ANALYSIS MODE



DETAILS OF HUMAN INPUTS TO FORCE AGENT

(ANALYST MODE)

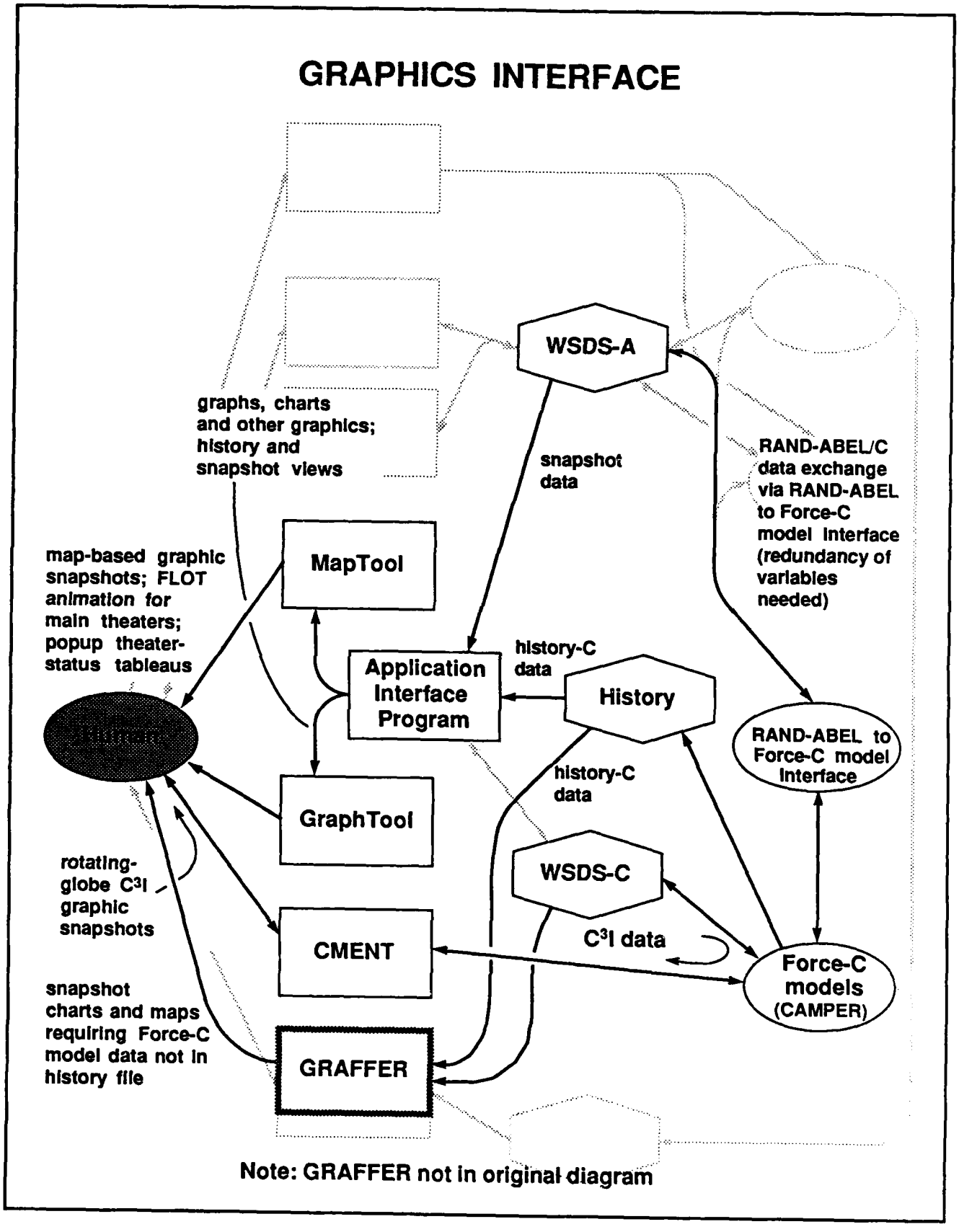
(Already discussed)

NOTES ON FORCE AGENT INTERFACES

PATH	TYPE OF DATA	RECIPIENT	VIA
1	Rules, including equations, in code	Force-A	Interpreter
2	Rules, including equations, in code	Control plan (supplementing Force-A)	Interpreter, Control Plan
3	Parameters, state variables, orders, and system params.	WSDS-A (for Force-A)	Interpreter, Control Plan
4	Parameters, state variables, orders, and system params.	WSDS-A (for Force-A)	Data Editor
5	Parameters, state variables, orders, and system params.	WSDS-A (for Force-A)	Data Editor and (Scenario Generator)
6	System parameters (e.g., log levels)	WSDS-A	Data Editor
7	Orders, parameters	WSDS-A, Force-A	CMENT
8	Orders, parameters, and system commands	WSDS-C, Force-C models	Interpreter, Control Plan
9	Orders, parameters, and system commands	WSDS-C, Force-C models	Data Editor, WSDS-A, Control (Scenario Gen.)
10	Orders, parameters, and system commands	WSDS-C, Force-C models	CMENT

NOTES ON FORCE AGENT INTERFACES

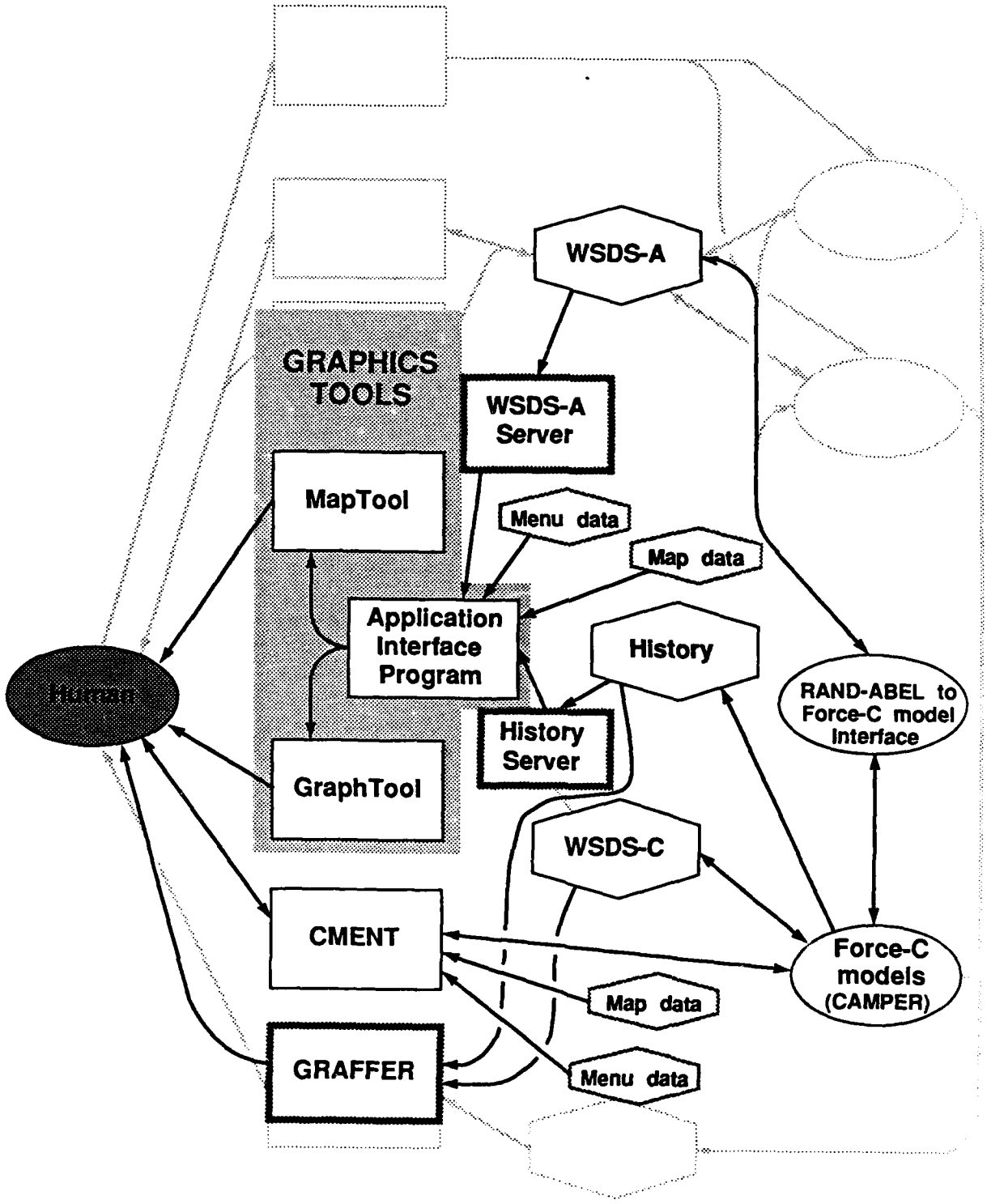
(Already discussed)


GRAPHICS INTERFACE
WSDS-A
graphs, charts and other graphics; history and snapshot views
snapshot data
RAND-ABEL/C data exchange via RAND-ABEL to Force-C model interface (redundancy of variables needed)
map-based graphic snapshots; FLOT animation for main theaters; popup theater-status tableaus
MapTool
Application Interface Program
history-C data
History
RAND-ABEL to Force-C model interface
GraphTool
history-C data
rotating-globe C3I graphic snapshots
WSDS-C
CMENT
C3I data
Force-C models (CAMPER)
snapshot charts and maps requiring Force-C model data not in history file
GRAFFER
Note: GRAFFER not in original diagram

GRAPHICS INTERFACE

Having discussed the various mechanisms for entering inputs, let us now look at outputs. This chart provides an overview of the graphics-output interface. It highlights the two tools of most interest, Graph Tool (developed by Larry Painter) and Map Tool (developed by Mark LaCasse). GRAFFER, shown here but not in previous charts, was a key element of the original RSAS but is being supplanted by these two tools; it still appears only because there remain a few graphics that still cannot be accessed by the newer tools. CMENT appears because it has two specialized graphical outputs of interest—rotating-globe displays of C^3I status and naval presence. Users not interested in strategic-nuclear operations can simplify their life (the RSAS is complex in part because it offers so many options) by focusing strictly on Graph Tool and Map Tool.

DETAILS OF GRAPHICS INTERFACE



DETAILS OF GRAPHICS INTERFACE

This chart introduces more details important to software specialists. The most significant point here is that good software design involves input-output "tools" that can be used in many different contexts with no new programming. For example, it makes no difference to the RSAS Graph Tool whether the data it receives come from model x or model y, or indeed whether the data even come from an RSAS model in the first place. If it did make a difference, it would mean that any time a model were changed or variables added the Tool would also have to be changed, creating a severe maintenance problem.

The general approach, then, is to insulate RSAS tools from the details of the models and data bases. Such specialized information is built into software entities such as the Applications Interface Program (AIP). The AIP can accept data from many different models, so long as those models transmit the data in a specific format, and so long as the AIP knows the names of the data elements.

To describe this in somewhat less abstract terms, consider for a moment a graph-drawing tool that takes input and plots graphs on a simple *x/y* display. The tool should only know that it is receiving pairs of data points; it should not care where the data come from or how they are generated. By contrast, one can imagine a graphical output program embedded within a complex model. This output program might have instructions causing it to take its first *x* value from a temporary storage location representing the output of a function that has just been called, or the current value of a pointer to a variable, and so on. Such a program could produce the graphics but would be extremely specialized, with the embedded code likely to obscure model operation as well.

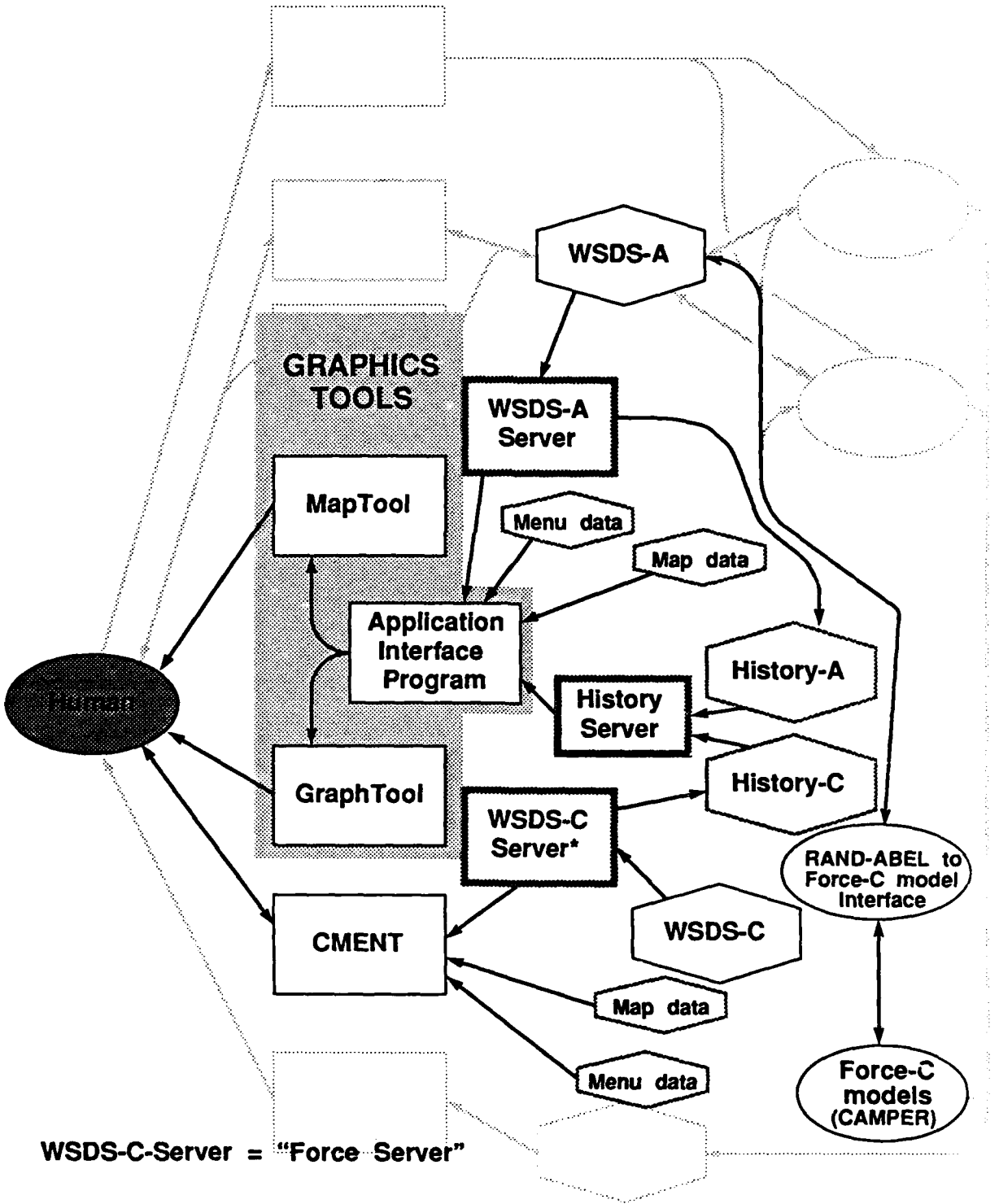
Another example may be useful. Imagine a graphics program that knows that the first *x* value is the 37th number in a particular file, the first *y* value is the 74th number, and so forth. It "knows" this only because the file has been carefully formatted and the graphics tool has been "hard-wired" for that format. If the model changes, the output file is likely to change as well (e.g., the first *x* value may now be the 39th number) and the graphics program will have to be reworked.

By contrast in the setup represented in this chart the user specifies by name the variables he wants to see plotted, and the AIP translates this request into a flow of data to the Graph Tool. Although some specialized knowledge concerning the interrelationships between variables is contained in the AIP (remember that Graph Tool itself is tasked only with plotting numbers regardless of origin), it still only refers to such variables by name, such that additions or deletions from the WSDS-A have no effect on it, and changes in the way that a variable is calculated generally leave the AIP unaffected as well.

The reason this is nontrivial is that complex programs such as the Force-C models (and portions of Force-A) can make extensive use of pointers and functions, and even though a variable may exist in *concept*, it may not be represented by an actual program variable—that is, there is no memory location associated with such an item. Instead, such a datum is generated as required; the output program must be able to ask things such as "Tell me the current value pointed to by ___ divided by the number generated by executing the function ___."

There are strong tensions between the desire for efficiency and immediate results on the one hand and software modularity and transparency on the other. In the original RSAS there was some tilting toward a "first and fastest" approach, but as the system matures we are moving toward greater modularity with its advantages of maintainability and clarity.

POSSIBLE GRAPHICS DESIGN FOR FUTURE RSAS



WSDS-C-Server = "Force Server"

POSSIBLE GRAPHICS DESIGN FOR FUTURE RSAS

This chart shows the data flow we hope to apply in future versions of the RSAS. GRAFFER will have disappeared and a new interface called the WSDS-C server will have emerged. In this system Graph Tool and Map Tool will have access to all Force and decision-model data via the WSDS-C and WSDS-A servers and the History Server, as coordinated through the AIP. As discussed above, the system will be quite modular, with changes to one component having minimal effect on the other components.

The practical significance of these improvements could be substantial for maintainability and simplicity as well as increasing the power of the interface as a whole. Many of the RAND-ABEL/C distinctions that currently cause trouble will no longer be discernible. Unfortunately, moving to such a server will require substantial reprogramming of Force-C, and thus might not come about for some time, if ever. In the meantime we are incrementally improving the efficiency of Force-C/RSAS-A communications without restructuring.

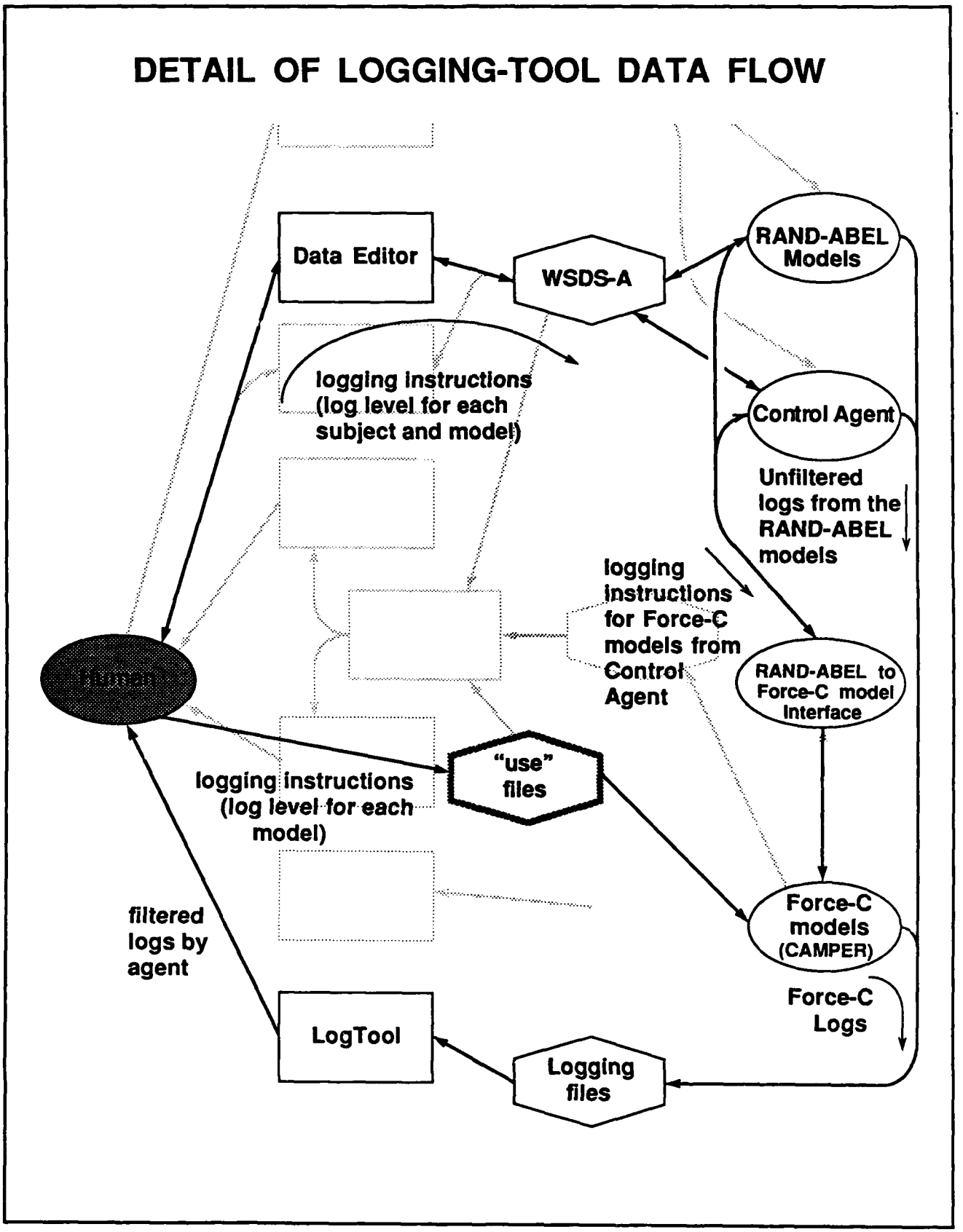
SOURCES OF GRAPHICS DATA IN THE RSAS

TYPE GRAPHIC	THEATER	ORIGIN OF DATA	GRAPHICS PROGRAM
History graphs of EDs, sorties, weapons, attrition, etc.	All	CAMPER and Force-A (mostly	Graph Tool (or GRAFFER)
Animated FLOT-map histories	Center Region or Korea	CAMPER	Map Tool/Graph Tool combination (or faster GRAFFER)
Most snapshot maps (LOCs, points, choke-points, FLOTs, forces, naval dispositions)	All	Various	Map Tool
Most bar-chart summaries (e.g., damage from nuclear attacks)	Intercon-tinental	CAMPER	Graph Tool
Conflict-state-by-theater graphic	Global	RAND-ABEL Models	Graph Tool
Conflict time line	Global	RAND-ABEL Models	Graph Tool
Force-distribution maps (e.g. worldwide distribution of US divisions	All	CAMPER	GRAFFER
Political cooperation and conflict-level maps	All	CAMPER	GRAFFER
SIOP/RISOP targeting and target statuses	Intercon-tinental	CAMPER	GRAFFER
Rotating globe C^3I node status	All	CAMPER	GRAFFER

SOURCES OF GRAPHIC DATA IN THE RSAS

This chart summarizes where the various graphics interfaces in the RSAS obtain their data. It will be of greatest use after the user is familiar with the displays available, at which point the chart may be useful in helping the user remember where the data displayed came from. For example, someone who has been using Graph Tool may wish at some point to see some strategic nuclear targeting displays that he "can't find" in the menus for Graph Tool. As shown on the chart, those displays are generated currently only by GRAFFER.

As a rule of thumb, Graph Tool and Map Tool are sufficient so long as the data come from the "History file" or from the WSDS-A. Not all Force-C data are maintained in the History file, however, because the file would become enormous. For example, current information on strategic nuclear targeting is data-intensive because there are a very large number of target types, delivery systems, and weapon allocations.


DETAIL OF LOGGING-TOOL DATA FLOW
Data Editor
WSDS-A
RAND-ABEL Models
logging instructions (log level for each subject and model)
Control Agent
Unfiltered logs from the RAND-ABEL models
logging instructions for Force-C models from Control Agent
Human
RAND-ABEL to Force-C model interface
"use" files
logging instructions (log level for each model)
filtered logs by agent
Force-C models (CAMPER)
Force-C Logs
LogTool
Logging files

DETAIL OF LOGGING-TOOL DATA FLOW

Here we have a more detailed representation of the RSAS logging system. The user can specify the level of information to be logged at the outset of the game, and can change that level at any time during the game for either C or RAND-ABEL variables.[1] However, there are additional options available for WSDS-A variables. In particular, the user can ask to see a file showing, say, only low-level Red Agent log data (roughly speaking, the Red Agent's decisions without explanations) even though the underlying log file contains data for Red, Blue, Green, and Force-A models in whatever maximum level of detail he specified earlier. These filtered logs are quite useful since the unfiltered log data is massive. At some point in the future the RSAS will have more extensive post-processing options for both the C and RAND-ABEL sides.

As a side note, although users of the RSAS have been very impressed with the information available, one of the relatively few criticisms of the system has been the lack of a Data Base Management System with which to restructure the output data. The Graph Tool provides substantial flexibility in creating new displays, but there is no capability in the RSAS to search out classes of data from the Force logs, for example, other than by the straightforward but inelegant procedure of doing text searches using a text editor or UNIX utility (we have used such procedures in RAND studies). It will not be practical to provide the additional capabilities until the WSDS-C server (Force server) exists since, once again, the key issue is having all important information represented as program variables.

---

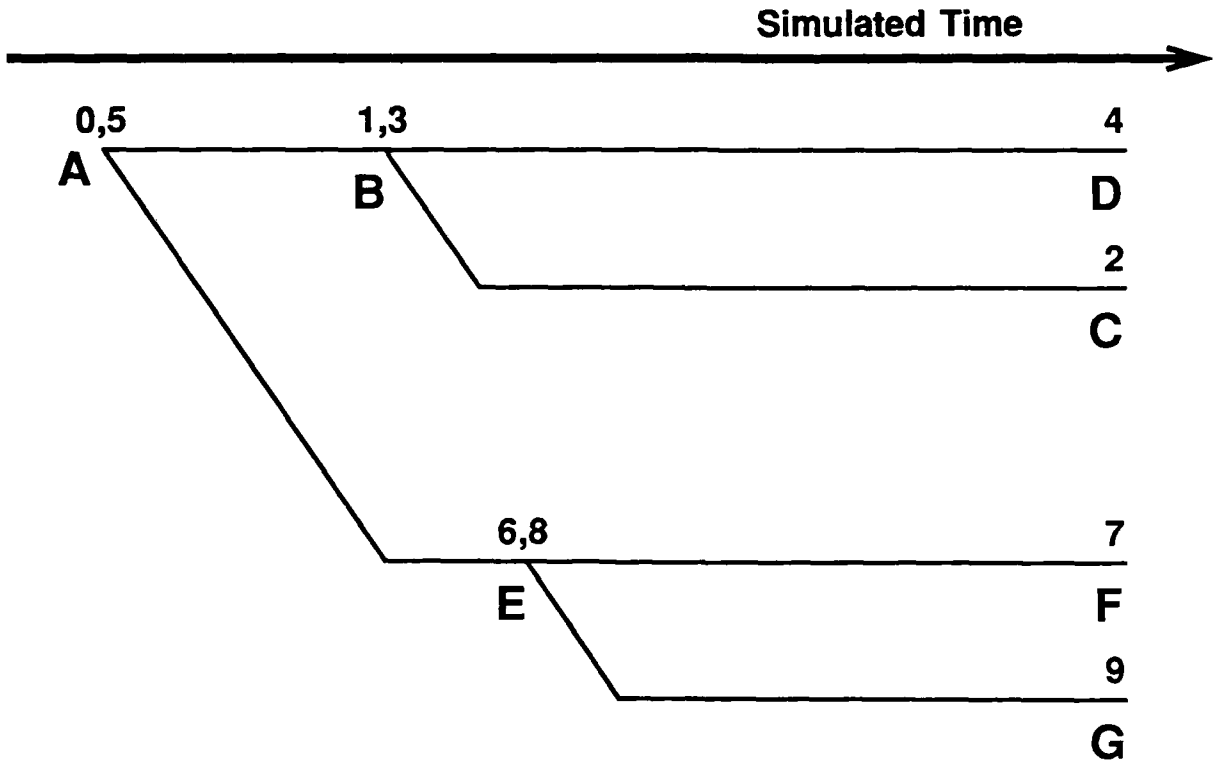
[1]In practice, users typically set Force-C log levels by editing the default "use" file in the Run directory, or by orders placed in Analyst Plans.

SPECIAL CAPABILITIES FOR SAVING AND BRANCHING

SPECIAL CAPABILITIES FOR SAVING AND BRANCHING

Having completed our discussion of data flow, let us now consider briefly some of the special features the RSAS has for doing excursions and what are called lookaheads.

ILLUSTRATIVE SAVING AND BRANCHING WITH RSAS



0. Read in initial WSDS, A. Start RSAS.
1. Save state B; make changes. Continue.
2. Save final state C.
3. Re-read state B. Continue.
4. Save final state D.
5. Start again with initial WSDS, A.
6. Save state E. Continue.
7. Save final state F.
8. Re-read state E; make changes.
9. Continue to end state G.

ILLUSTRATIVE SAVING AND BRANCHING WITH THE RSAS

In this Note we don't say much about the saving and branching features of the RSAS, but they are both novel and powerful. This chart illustrates some of the many capabilities they provide. Note that each of the saved states can be used as the start of another session—one does not have to rerun from the beginning.

One capability not illustrated here is the ability to store only the changes in the WSDS-A's state relative to a given baseline; the file so produced is called a "delta-WSDS-A." Use of delta-WSDS-As can greatly speed multicase analysis, especially when the Data Editor's ability to produce delta-WSDS-As directly is exploited. Frequently, a given set of changes to the RSAS simulation's initial conditions is stored as a delta-WSDS-A, which can then be combined with other delta-WSDS-As and used to initialize a particular case. Unfortunately, because WSDS-C data cannot be stored in a delta-WSDS-A, the usefulness of the approach is limited to initial states. In practice, then, one uses a combination of delta-WSDSs and Analyst Plans or scenario-generator tableaus.

CONTENTS

CONTROL FLOW

In the 1980s it is generally becoming understood that the complex flow charts of yesteryear are not very useful despite the enormous effort required to produce them. Data-flow diagrams are generally more informative and to-the-point. However, certain aspects of control flow are very important since they relate to finer details that influence the fidelity of the models themselves—data-flow diagrams can leave ambiguity in temporal aspects of the system. Also, we wish to show programmers how the various control-flow mechanisms have been implemented in the RSAS.

RSAS CONTROL FLOW

- The Force-C models control simulation clock with variable time step
- Decision models act in frozen time
- Order of agent moves depends on agent wakeup rules and polling sequence
- After each clock advance, Monitor tests for wakeup requests
- Polling order breaks ties on which agent moves next
- Models may act several times during frozen time, reacting to others moves until decisions are stable
- Simulation is not "ping-pong" decisionmaking; can reflect delays in decisions and information
- During lookahead, simulation is in "lookahead time"

RSAS CONTROL FLOW

This chart summarizes some of the more important aspects of RSAS control flow. Most of the comments need no elaboration, but we should emphasize that the RSAS does not have a limitation that many observers assumed it had based upon earlier diagrams of its control flow. In particular, the RSAS does not assume a simple ping-pong game between Red and Blue. Red may be acting at a time when it does not yet know some of what Blue has done or what Blue has already decided to do. In the real world there are delays and limitations of information—Red most definitely does not know what Blue has decided, but rather can only see some of the results of Blue's decisions upon the physical world (including diplomatic messages directed to Red). Red's inferences may be wrong.

In a few charts we shall show a picture of how control flow operates; this may make some of the concepts described here a bit more clear.

POLLING ORDER IN RSAS CONTROL FLOW

- Default polling order:
  - — Parent, then children (and children's children, etc.), then parent's siblings
  - — After wakeup, polling resumes at top-level coprocess; after last top-level coprocess start over
  - — If no one wakes up for full cycle, polling ends (and Force Agent is run, advancing clock)
- Order of top-level coprocesses: Force-A, Control, Green, Blue, and Red
- Order of lower-level coprocesses (e.g., SCLs and ACLs) is variable (see source code)
- System Monitor controls flow of execution; may also have own wakeup rules, polled after Force Agent but before Control Agent
- User may use tool displaying coprocess hierarchy (H-Tool) to poll or execute coprocess out of sequence

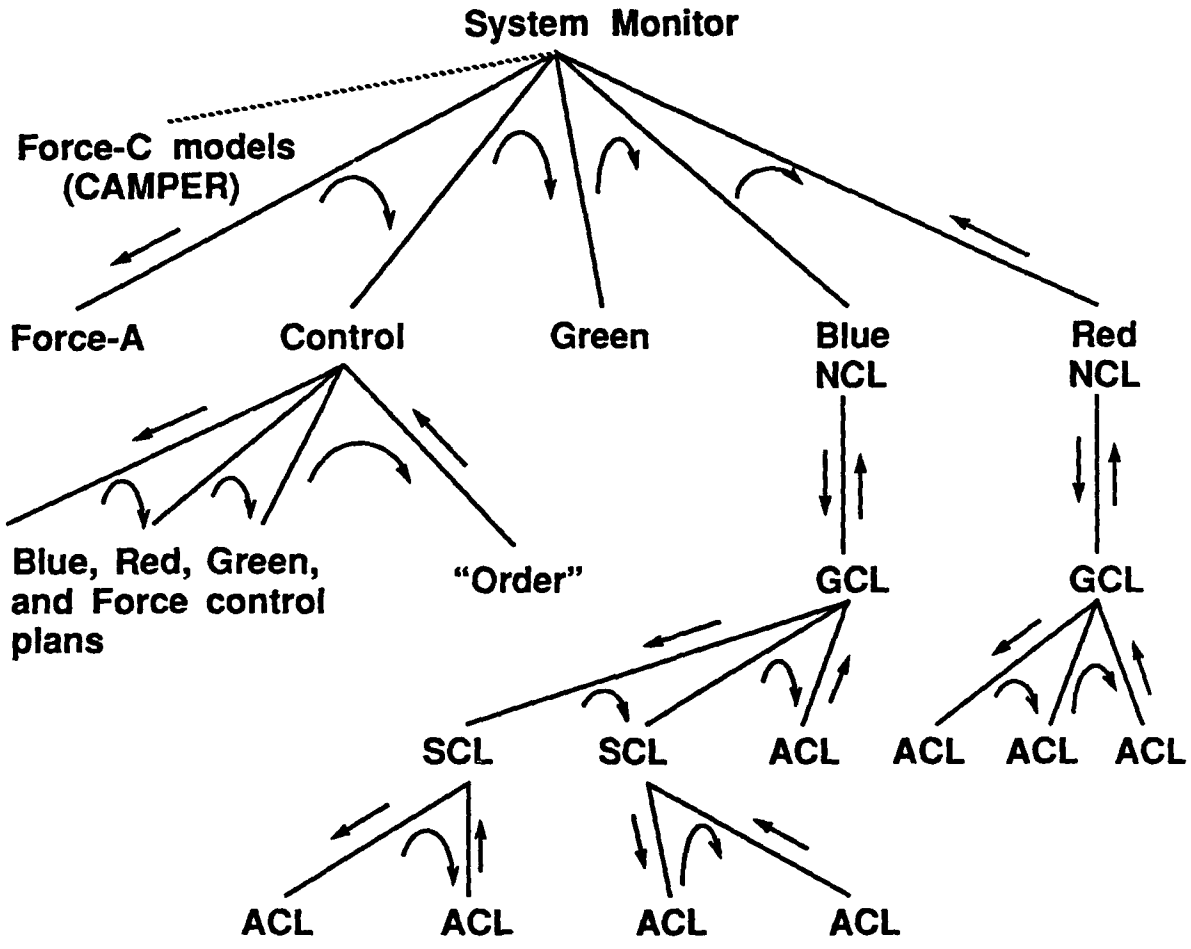
NOTE: except for Force-A the order of coprocess polling "should not" matter if rules are robust.

POLLING ORDER IN RSAS CONTROL FLOW

As noted above, there has to be a polling-order logic to resolve which agent moves next when two or more agents both want to move. This logic can be overridden by using the Hierarchy Tool or H-Tool developed by colleague Mark LaCasse, but this is uncommon; the baseline logic is shown in this chart.

The next chart, POLLING THE TREE OF RSAS COPROCESSES, will make this easier to understand.

POLLING THE TREE OF RSAS COPROCESSES



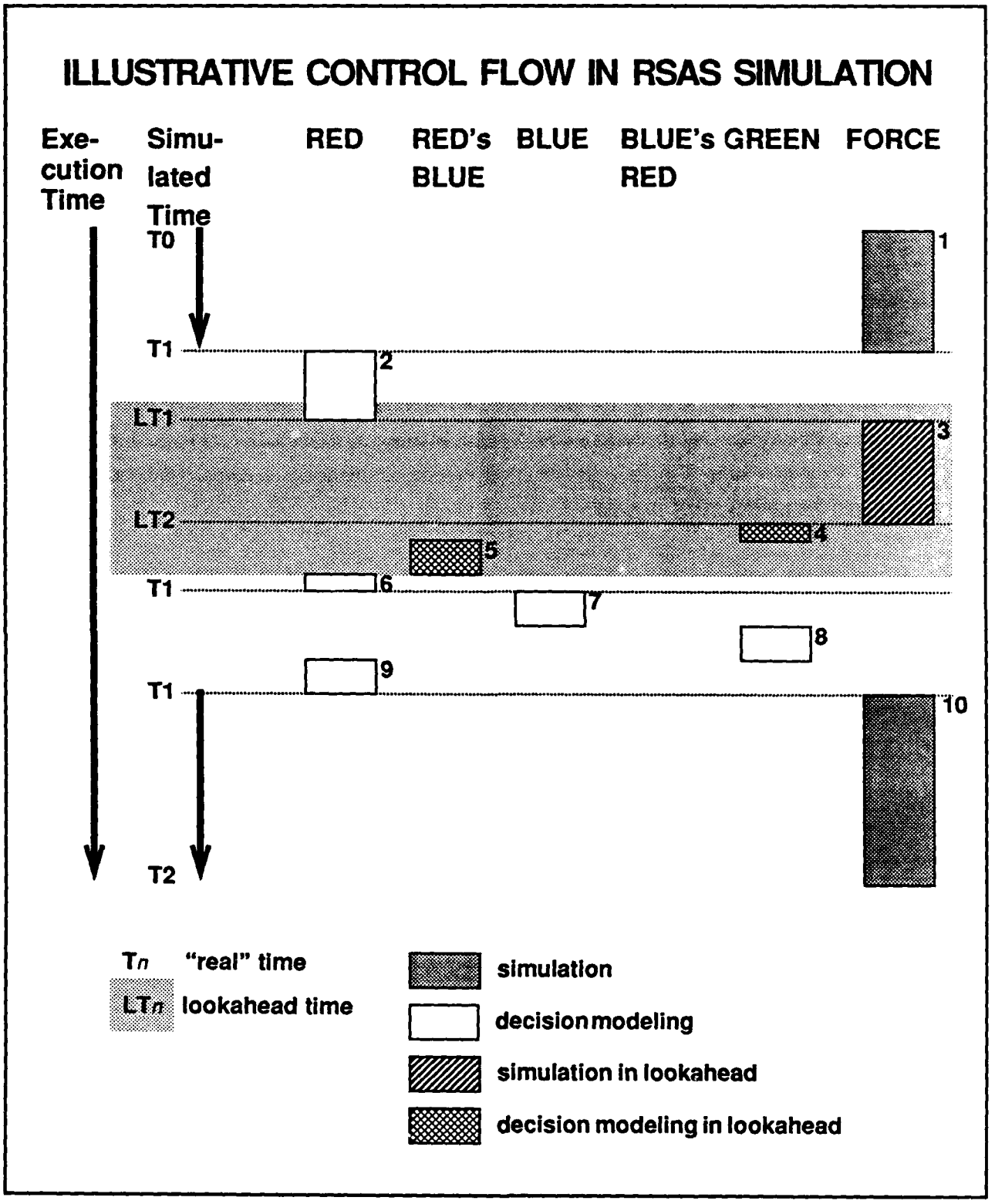
Illustrative only.
(Blue and Red may have arbitrary number of SCL and ACL coprocesses; Force-A, etc., may have child coprocesses.)

..... Force-C models (CAMPER) and System Monitor are both UNIX processes and can run on separate machines.

POLLING THE TREE OF RSAS COPROCESSES

As shown here pictorially, System Monitor is the ultimate "parent" coprocess. At the next level there are six "siblings" (the Force-C models, Force-A, and so on), most of which have "children." The Force-C models are contained in a separate UNIX process from System Monitor and can therefore be run on a separate CPU if desired. In the current system the other agents shown are all part of the same UNIX process as System Monitor.

Polling takes place in a depth-first manner—that is, after a given coprocess is polled, its children (if any) are polled before its siblings (if any), and the same is done for any children of its children, and so on. If any wakeup rules for a polled coprocess indicate that it should be awakened, that coprocess is run until its next *Sleep* request, after which polling resumes at the relevant top-level coprocess (e.g. the Blue NCL is polled after any of the Blue coprocesses—its descendents—awakens) and continues down the tree. Only when all coprocesses have declined to awaken does System Monitor turn control over to the Force-C models. Then the Force-C models perform their combat simulation and advance the game clock, and the polling cycle repeats.


ILLUSTRATIVE CONTROL FLOW IN RSAS SIMULATION
Exe-cution Time
Simu-lated Time
RED
RED's BLUE
BLUE
BLUE's RED
GREEN FORCE
T0
T1
LT1
LT2
T1
T1
T2
1
2
3
4
5
6
7
8
9
10
Tn "real" time
LTn lookahead time
simulation
decision modeling
simulation in lookahead
decision modeling in lookahead

ILLUSTRATIVE CONTROL FLOW IN RSAS SIMULATION

This chart shows pictorially how control shifts in the process of several agent moves, including moves occurring "in lookahead." The next chart describes the meaning of the steps in more detail, but basically what is being depicted here is a slice out of a game. At the beginning (**1**), Force Agent is updating the state of the simulated world—computing the result of battles and so on. It advances the clock to time **T1** and relinquishes control. Red wakes up (**2**) and begins a decision process requiring a lookahead simulation. This includes some simulation by Force (**3**) and decisions by Red's models of Green (**4**, using Red's parameter values for the Green models) and Blue (**5**). After the lookahead, Red completes its decisions(**6**), Blue moves(**7**), Green moves (**8**) and then Red moves again(**9**). Force Agent then regains control (**10**), still at simulated time **T1**, and the game continues.

The Force Agent's time steps depend upon circumstances. The nominal time step is four hours, but it becomes much shorter if there is nuclear use, and even shorter (a few minutes) when Red or Blue has tactical warning of an incoming attack on its homeland. The user or other agents can also request a shorter time step.

ILLUSTRATIVE CONTROL FLOW (CONT'D)

EVENT	ACTIVITY
1	Simulation of events in physical world between simulated time T0 to T1
2	Decisionmaking at T1 by Red; Red decides to do lookahead
3	Red decisionmaking continues: simulation of events starts at lookahead time LT1
4	Red decisionmaking continues: Green Agent, adjusted to Red's assumptions, does decisionmaking in Red's lookahead at time LT2
5	Red decisionmaking continues: Red's Blue does decisionmaking in lookahead at time LT2
6	Lookahead ends as Red concludes decisionmaking by evaluating lookahead's results
7	Blue decisionmaking; we are again in "real time" at time T1
8	Green decisionmaking at time T1
9	More Red decisionmaking, still at time T1
10	Simulation of events in physical world between simulated time T1 to T2

ILLUSTRATIVE CONTROL FLOW (CONT'D)

(Already discussed)

CONTENTS

CONTENTS

We shall now discuss certain aspects of the man-machine interface briefly, although we clearly have touched on many relevant issues already.

MAN-MACHINE ENVIRONMENT

- Sun workstations (graphics, windows, networking, ownership...)
- RAND-ABEL language
- On-line documentation for many models and tools
- RSAS tools

MAN-MACHINE ENVIRONMENT

This chart shows the elements of the RSAS environment, but we will discuss these elements here only briefly because they need to be examined first-hand and in most cases are documented separately and in much more detail than possible here. The next chart lists the various RSAS tools, but does not discuss them in detail. See the *Beginner's Manual* (Gillogly, furnished to RSAS sites) and a paper describing the explanation mechanisms of the RSAS (Davis, 1988).

RSAS TOOLS

- Interpreter: change RAND-ABEL interactively
- Data Editor: view and change RAND-ABEL variables
- H-Tool: monitor control flow, manually poll or execute agents, stop play at chosen point for human decisions
- Cross-referencing Tool: declarations and ranges of RAND-ABEL variables and functions
- Logging Tool: provide variable-resolution reports
- Source Code Menu Tool: select relevant portion of source code
- CMENT provide nested-menu interface to Force-C models (CAMPER)
- Graphics Tools: maps, history graphs, bar graphs, and other displays; tools for constructing line and bar graphs

RSAS TOOLS

This chart lists the most important RSAS tools, all of which are application independent. We have discussed most of them already, but let us make a few review comments here.

As a practical matter, being able to change parameters and rules interactively by using the interpreter, data editor, and CMENT, and being able to review simulation results and diagnose problems with convenient graphical tools and logging, have proven essential in making it attractive for *analysts* to work directly with the RSAS at the work station—using the system during their creative activities as well as for production. This flexibility is essential to our concept of what a modern knowledge-based simulation should be.

There are several more minor tools not discussed earlier. One of these is the Hierarchy Tool (H-Tool), developed by Mark LaCasse. It displays the hierarchy of coprocesses and highlights the one currently active. In addition, the user can select one of the H-Tool "bubbles" and specify that the game suspend activity when the turn of that object comes up, allowing him to examine or modify the simulation. H-Tool also gives a list of a coprocess' current wakeup rules, and allows a coprocess to be awakened out of turn.

Another useful tool is the Sourcecode Menu Tool, which has proven quite helpful in dealing with the hierarchy of sourcecode. Developed by Robert Weissler, this tool consists of nested menus keyed to the directory and file structure of the RAND-ABEL source code. Since this structure is identical to the conceptual structure of the RSAS RAND-ABEL models, the user can easily find model components and learn how they are fitted together. By selecting a menu item, he can bring up the rule set corresponding to a particular code module. This facility also allows part or all of the source code hierarchy to be searched for a given variable or function name or other code fragment.

The Cross Referencing Tool, developed by Sharyne Blixt, is tied to a data base that cross-indexes the various functions, variables, and values of the RAND-ABEL source code. Someone writing new RAND-ABEL rules for a given model who has forgotten the permitted values (or more important, the correct spelling) of existing qualitative variables (e.g., the values of the variable Strategic-warning or Global-objectives) can use the Cross Referencing Tool to quickly obtain definitive information on such matters. The Cross Referencing Tool also allows direct recall of a variable or function declaration in the Data Dictionary; this is quite useful since the comments there often help explain the meaning of qualitative variables.

CONTENTS

CONTENTS

Next, let us discuss the current status of the RSAS as it appears to the authors. Our assessment is clearly subjective, but may be useful nonetheless. We shall focus here on problems, even though we are quite pleased with the current RSAS and its usefulness as it stands.

LIMITATIONS AND SHORTCOMINGS

- Interfaces to Force-C models depend on details of Force-C internal programming (i.e., the Force-C models and their components are not software modules)

  Result: inefficient communications between C and RAND-ABEL programs, degraded performance, complexity, maintenance problems
- The RSAS's extreme flexibility can be paralyzing: "too many options;" need to identify narrow, tailored, and simple user modes
- Until completion of the Source Code Data Base facility (late 1988 or early 1989) recompilation of RAND-ABEL programs involving new variables will be very slow
- Changing C programs or displays, or verifying model details, requires expertise in C/UNIX and the RSAS
- Numerous bugs likely in models not yet used in serious study efforts (much of the RSAS)
- Force-C logs and displays inadequate for simple calculational audits; RAND-ABEL models better, but have room for improvement
- Significant differences exist between C- and RAND-ABEL-side interfaces, increasing cognitive burden for user
- WSDS saving and reading mechanisms appear fragile, with occasional "no-reason" failures; re-implementation of coprocess save mechanism will eventually be desired
- Various features not yet implemented: e.g., much of CMENT's interfaces and full-feature Cross Referencing Tool
- Non-uniform HELP procedures
- Some user interfaces not fully documented
- RAND-ABEL recompilations can be complicated

LIMITATIONS AND SHORTCOMINGS

In this chart we summarize the problems we know exist and consider the most important. Little discussion is necessary.

By far the most serious problems relate to the interface between RAND-ABEL and C models. Some of the very features that make C powerful and efficient for the global RSAS simulation (e.g., untyped pointers) make interfacing to C data difficult. We recognized that this would be the case, but were unable to move toward a more fully integrated system until completion of RSAS 3.0.

SELECTED COMMENTS ON RSAS AS SOFTWARE

- Extremely interactive and flexible, but correspondingly complex
- Unique worldwide for complexity management in very large knowledge-based simulation
- Models have more than 150,000 lines each of RAND-ABEL and C (RAND-ABEL part translates into about 450,000 lines of C); interfaces and other supporting software approximately double the total figure
- Operates on Berkeley 4.x UNIX systems with 68000-series CPU; other UNIXes or CPUs would require a moderate amount of model and non-interface support software reprogramming
- Exploits Sun workstation tools. Reprogramming of interface would be necessary for other hardware
- Minimum hardware: 12MB RAM, 60MB virtual address space, 280MB mass storage

SELECTED COMMENTS ON RSAS AS SOFTWARE

The first thing to be said in a net assessment of the RSAS as software is that it is extremely interactive and flexible, as intended. It is also *very* complex. To say the RSAS is user-friendly as of August 1988 would be a significant exaggeration, because the substantive content and sophistication of the RSAS has increased rapidly as the techniques for comprehending complex interactions have improved. Users of the RSAS routinely do things (with some difficulty) that would not even have been *attempted* previously.

So far as we know, there is no comparable system anywhere in the world. There are rich literatures on expert systems and simulation, but knowledge-based simulation is new and the other systems of which we are aware have been much less ambitious. The methods used in the majority of academic studies of related technology (e.g., expert systems) simply cannot cope well with very large and complex systems in which there are thousands of rules and large numbers of interacting objects.

Measuring programs is notoriously difficult, but using the most common measure, the RSAS models have more than 150,000 lines each of C and RAND-ABEL. However, our extensive use of RAND-ABEL tables, which are a particularly dense form of code, means the RAND-ABEL part of the RSAS translates into about 450,000 lines of C. The run time for the RSAS on a Sun 3/60 workstation varies from tens of seconds to a few minutes per day of simulated crisis and conflict, depending on how many theaters are being simulated, whether the political decision models are being employed, log levels, and so on. A global war game can be conducted in about 1-3 hours. Analysts can usually diagnose results (except for problems due to bugs in the C programs) within minutes—or perhaps tens of minutes in bad cases. This may require consultation with other analysts or programmers, however, because none of us currently are expert in all the various models and data bases—in terms of either substance or software. Thus for all but highly focused applications of the RSAS, effective use demands having a team of analysts and programmers.

The RSAS is a large collection of interacting software, and to use it effectively we strongly recommend that users have at minimum 12 MB of physical memory, at least 60 MB of virtual memory, and 280 MB of mass storage. Although the RSAS will run on smaller systems, performance penalties can be severe and can substantially reduce the productivity and ease-of-use of the system. The issue here is not just speed, but also the ability to use multiple tools simultaneously, record intermediate world states and come back to them, store multiple history files and logs, and so on. In this day and age it is unreasonable to have analysts wasting their time worrying over storage and memory use when hardware costs are far lower than analyst costs.

CONTENTS

CONTENTS

It seems appropriate to conclude with a list of references for various aspects of the RSAS; also, following the appendix there is an annotated bibliography and some comments on which references are the most useful.

REFERENCES

MODELS

- Existing references somewhat outdated, but useful for background. See Davis and Winnefeld (1983)
- Davis, Bankes, and Kahan (1986); Davis (1985)
- For discussion of Main Theater Force models, see Bennett, Jones, Bullock, and Davis (forthcoming)

RELATIONSHIP TO ARTIFICIAL INTELLIGENCE

- See Davis (1985)

RAND-ABEL LANGUAGE

- For rationale, see Shapiro, Hall, Anderson, and LaCasse (1985)
- For reference manual, see Shapiro, et al. (1988)

RSAS ENVIRONMENT

- For a beginner's guide, see Gillogly (furnished to RSAS sites)
- For RAND-ABEL model explanation mechanisms, see Davis (1988)

TOOLS AND INTERFACES

- See on-line documentation (under main menu item Doc)

Appendix

TERMINOLOGICAL EQUIVALENCES

Ignoring the distinction between models and programs, the following equivalences hold:

RSAS-C = CAMPER

Force-C = CAMPER + related data bases and tools

Force-A = S-LAND's referee model + flag model + other

Force Agent = CAMPAIGN

CAMPAIGN = CAMPAIGN-MT + CAMPAIGN-ALT + other force models

CAMPAIGN = CAMPAIGN-MT + CAMPAIGN-ALT (S-LAND)+ other force models

Common incorrect (i.e. obsolete) terminology one may hear or see in older publications:

- "Force" to mean "Force-C" or the Force-C models
- "CAMPAIGN" to mean "CAMPAIGN-MT"
- "WSDS" to mean "WSDS-A"
- "Force Server" to mean the interface between Force-C and the RAND-ABEL models. The interface is not a true data server.
- "Agents" excluding Force-C models (i.e. failing to include non-RAND-ABEL entities)
- "Agents" to mean those models not part of CAMPAIGN

ANNOTATED BIBLIOGRAPHY

Allen, Patrick D., and Barry A. Wilson, *Secondary Land Theater Model*, The RAND Corporation, N-2625-NA, July 1987. The S-Land model exploits the RAND-ABEL language in making it possible to develop theater-level models quickly. It can be used also to develop specialized models for main theaters.

Bennett, Bruce W., Carl M. Jones, Arthur M. Bullock, and Paul K. Davis, *Main Theater Warfare Modeling in the RAND Strategy Assessment System*, The RAND Corporation, N-2743-NA , forthcoming. A fairly detailed overview of the models used primarily for Europe's Central Region.

Davis, Paul K., *Applying Artificial Intelligence Techniques to Strategic-Level Gaming and Simulation*, The RAND Corporation, P-7120, November 1985 (reprinted as N-2752-RC, March 1988). Published also as a chapter in M. S. Elzas, T. I. Oren, and B. P. Zeigler (eds.), *Modeling and Simulation Methodology in the Artificial Intelligence Era*, Elsevier Science Publishers B. V., North-Holland, 1986.

Davis, Paul K., *Explanation Mechanisms for Knowledge-Based Models in the RAND Strategy Assessment System* , The RAND Corporation, N-2711-NA, August 1988.

Davis, Paul K., *A New Analytic Technique for the Study of Deterrence, Escalation Control, and War Termination*, The RAND Corporation, P-7224, May 1986. Also published as a chapter in Stephen J. Cimbala (ed.), *Artificial Intelligence and National Security*, Lexington Books, D.C. Heath and Company, Lexington, MA 1987. Describes applications of NCL models.

Davis, Paul K., Steven C. Bankes, and James P. Kahan, *A New Methodology for Modeling National Command Level Decisionmaking in War Games and Simulations*, The RAND Corporation, R-3290-NA, July 1986.

Davis, Paul K. and James A. Winnefeld, *The RAND Strategy Assessment Center: An Overview and Interim Conclusions about Potential Utility and Development Options*, The RAND Corporation, R-2945-DNA, March 1983. Now obsolete except for background on original concepts.

Gillogly, Marrietta, *A Beginner's Manual for the RAND Strategy Assessment System*, an unpublished draft manual available only to RSAS government agencies with the RSAS.

Goldberg, Adele, and Alan Kay, *Smalltalk-72 Instruction Manual,* SSL 76-6, Xerox PARC, Palo Alto, CA, 1976.

Graubard, Morlie H., and Carl H. Builder, *RAND's Strategic Assessment Center: An Overview of the Concept,* The RAND Corporation, N-1583-DNA, September 1980.

Knuth, Donald, *The Art of Computer Programming,* 2nd ed., Addison Wesley Publishing Co., Reading, MA, 1973.

Klahr, Philip, and W. S. Faught, "Knowledge-Based Simulation," Proceedings of the First Annual National Conference on Artificial Intelligence, Palo Alto, 1980. One of the earliest references to knowledge-based simulation based on work at RAND.

LaCasse, Jean, *Reference Manual for the RSAS Data Editor,* The RAND Corporation, forthcoming.

LaCasse, Jean, *Defining Displays for the RSAS Data Editor,* The RAND Corporation, forthcoming.

McArthur, D., and Philip Klahr, *The ROSS Language Manual,* The RAND Corporation, N-1854-AF, September 1982.

Oren, Tuncer, and Bernard P. Zeigler, "Artificial Intelligence in Modeling and Simulation: Directions to Explore," *Simulation,* Vol. 48, No. 4, April 1987. Summarizes major directions briefly and provides guide to recent academic literature.

Reddy, Ramana, "Epistemology of Knowledge Based Simulation," *Simulation,* Vol. 48, No. 4, April 1987. A good general discussion; also references Reddy's and colleagues' early (circa 1981) work on the subject.

Schwabe, William, *Strategic Analysis as Though Nonsuperpowers Matter,* The RAND Corporation, N-1997-NDNA, June 1983 (a dissertation for the RAND Graduate School, this describes the motivation behind Green Agent, then called Scenario Agent).

Shapiro, Norman, H. Edward Hall, Robert Anderson, Mark LaCasse, Marrietta Gillogly, and Robert Weissler, *The RAND-ABEL Programming Language: Reference Manual,* The RAND Corporation, N-2367-1-NA, forthcoming (this is a revision of a 1985 document).

Shapiro, Norman, H. Edward Hall, Robert Anderson, and Mark LaCasse, *The RAND-ABEL Programming Language: History, Rationale and Design,* The RAND Corporation, R-3274-NA August 1985.

Shlapak, David A., William L. Schwabe, Mark A. Lorell, and Yoav Ben-Horin, *The RAND Strategy Assessment System's Green Agent Model of Third-Country Behavior in Superpower Crises and Conflict,* The RAND Corporation, N-2363-1-NA, September 1986. See also the 1988 user's manual for the Green Agent written by David Shlapak (an unpublished draft available only to government agencies possessing the RSAS).

Steeb, Randall, and James Gillogly, *Design for an Advanced Red Agent for the RAND Strategy Assessment Center,* The RAND Corporation, R-2977-DNA, 1983.

Zeigler, Barnard P., "Toward a Simulation Methodology for Variable Structure Modeling," in M. S. Elzas, T. I. Oren, and B. P. Zeigler (eds.), *Modeling and Simulation Methodology in the Artificial Intelligence Era,* Elsevier Science Publishers B. V., North Holland, 1986.